ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Zdroba Dániel**
2023	Hallgató törzskönyvi száma:	**T/006439/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Kiberfizikai Rendszerek Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Zdroba Dániel**
Törzskönyvi száma: T/006439/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Natív mobil és asztali alkalmazás fejlesztése távoli elérésű forgógép védőmodulhoz
Native mobile and desktop application development for a remote access rotary machine protection module

Intézményi konzulens: Sziklai Zsolt
Külső konzulens: Fésüs Péter

Beadási határidő: 2022. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
IoT, beágyazott rendszerek és robotika specializáció

A feladat

Tervezzen és készítsen egy olyan alkalmazást, ami forgógép védőmodulok távoli eléréséhez használható. A fejlesztés legyen platformfüggetlen, az elkészült program legyen használható iOS, Android és UWP platformokon.
Az alkalmazás az MQTT protokollban definiált kliens szerepét töltse be, legyen képes üzenetek publikálására, topic-okra való feliratkozásra, és tudjon csatlakozni ilyen kommunikációs protokollt használó hálózathoz. Legyen képes listázni az aktuálisan figyelt modulokat, valamint tudjon a felvenni továbbiakat, vagy törölni modult a figyelt elemek gyűjteményéből. Továbbá legyen képes a listázott modulok figyelésének ki- és bekapcsolására.
Jelenítse meg a kiválasztott modul által mért adatokat, valamint adjon lehetőséget a motor ki- és bekapcsolására. A rendszer naplózza a beérkező adatokat modulok szerint és tegye lehetővé a naplózott / mért összes adat megjelenítését.
A program inaktív állapotban a figyelje a modulok beérkező adatait. Abban az esetben, ha az ilyenkor beérkezett adat nem egy előre beállított felső és alsó határ közé esik, küldjön riasztást.

A dolgozatnak tartalmaznia kell:

- a megoldandó feladat pontos leírását és részletes elemzését,
- a lehetséges megvalósítási módokat és megoldásokat,
- a részletes rendszertervet és annak indoklását,
- a megvalósítás során használt technológiák bemutatását,
- a megvalósítás menetét, a munkafázisok és a tapasztalatok leírását,
- a tesztelési módszereket, tesztelési terveket és a tesztelés eredményeit,
- az elkészült rendszer alkalmazási, és továbbfejlesztési lehetőségeit,
- online a kifejlesztett rendszer forráskódját, illetve a készített dokumentációkat.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

Ph.

.....
Dr. Fleiner Rita
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. december 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 saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022. 12.07.

Zdroba Dániel

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Zdroba Dániel

Neptun Kód: [redacted]

Tagozat: nappali

Telefon: [redacted]

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

Szakdolgozat / Diplomamunka[1] címe magyarul:
Natív mobil és asztali alkalmazás fejlesztése távoli elérésű forgógép védőmodulhoz

Szakdolgozat / Diplomamunka[2] címe angolul:
Native mobile and desktop application development for a remote access rotary machine protection module …….....

Intézményi konzulens: Fésüs Péter .....

Külső konzulens: .....

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.02.17.	Szakdolgozat felépítéséhez tanácsadás.	Fésüs Péter
2.	2022.03.03.	Feladatlap áttekintése, leadási határidejének megbeszélése.	Fésüs Péter
3.	2022.04.19.	Első vázlat tartalmi, formai áttekintése, tanácsadás.	Fésüs Péter
4.	2022.04.28.	Leadási határidő, ahhoz szükséges dokumentumok, prezentáció szerkezeti követelményeinek megbeszélése.	Fésüs Péter
5.	2022.05.03	Közel végleges dokumentum tartalmi és formai áttekintése, tanácsadás.	Fésüs Péter

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplomamunka 2 / Diplomamunka 3 / Diplomamunka 4 [3] tantárgy követelményét teljesítette, beszámolóra / védésre [4]bocsátható.

Fésüs Péter
.....

Intézményi konzulens

Budapest, 2022.05.06.....

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Zdroba Dániel

Neptun Kód: [redacted]

Tagozat: nappali

Telefon: [redacted]

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

Szakdolgozat / Diplomamunka[1] címe magyarul:

Natív mobil és asztali alkalmazás fejlesztése távoli elérésű forgógép védőmodulhoz

Szakdolgozat / Diplomamunka[2] címe angolul:

Native mobile and desktop application development for a remote access rotary machine protection module

Intézményi konzulens: Sziklai Zsolt

Külső konzulens: Fésüs Péter

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.11.03.	Dokumentum második felének felépítése, beszámolás a megvalósításról	
2.	2022.11.17.	Kérdések a feladatlapról, konzultációs naplóról és a tesztelésről	
3.	2022.11.24.	Ábra készítési kérdés és adminisztráció	
4.	2022.12.01.	Tartalmi kérdés és véleményezés	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat / Szakdolgozat I. / Szakdolgozat II. / Diplomamunka II. / Diplomamunka III. / Diplomamunka IV. / Projeklabor 2. / Projektlabor 3. / Záródolgozati projekt [3] tantárgy követelményét teljesítette, beszámolóra / védésre [4]bocsátható.

Javasolt érdemjegy: .....

.....

Intézményi konzulens

Budapest, 2022. 12. 08.

---

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

Tartalomjegyzék

1 Bevezetés 4
2 A probléma megfogalmazása 6
2.1 Program feladatai 6
2.1.1 Megfigyelt védőmodulok kezelése 6
2.1.2 Védőmodul adatainak leolvasása 6
2.1.3 Távvezérlés 6
2.1.4 Adatok naplózása 7
2.1.5 Felhasználó értesítése 7
2.2 Feladatok összegzése, és a megoldandó probléma 7
3 Irodalomkutatás 9
3.1 Internet of Things 9
3.1.1 Távoli elérésű motorvédő modulok állapota 10
3.2 MQTT lapú kommunikáció 11
3.2.1 Működése 12
3.2.2 MQTT a rendszerben 12
3.3 Platformfüggetlen alkalmazás fejlesztés 13
3.3.1 Célja 14
3.3.2 Előnyei, hátrányai 14
3.3.3 Xamarin 15
3.3.4 React Native 16
3.4 Értesítési rendszerek 17
3.4.1 E-mail 17
3.4.2 Push értesítés, lokális és távoli 18
3.5 A megoldási módszerek kiválasztása, ezek indoklása 18
3.5.1 Xamarin 19
3.5.2 Lokális push értesítések 20
4 Igényspecifikáció 21
4.1 Funkciók és menürendszer 21
4.2 Adatok tárolása 23
4.3 Kommunikáció 23

5 Tervezési munkafázisok, tapasztalatok és a rendszerterv 25
5.1 Tervezési fázisok és tapasztalatok 25
5.2 Rendszerterv 26
5.2.1 Architektúra 26
5.2.2 Adat szint 27
5.2.3 Kommunikációs egység 28
5.2.4 Menürendszer 29
5.2.5 Értesítéskezelő egység 29
5.2.6 Platformspecifikus megoldások 30
6 Megvalósítás 31
6.1 Fejlesztési környezet 31
6.2 Adat szint 31
6.3 Kommunikációs egység 32
6.4 Menürendszer 33
6.4.1 Felépítés és navigálás 33
6.4.2 Működése 36
6.5 Értesítéskezelő egység 37
6.6 Egységek kommunikációja 38
6.7 Platformspecifikus megoldások 39
6.7.1 Android 39
6.7.2 iOS 40
7 Tesztelés 41
7.1 Felhasznált eszközök 41
7.2 Tesztelt körülmények, feladatok és folyamatok 41
8 Eredmények és továbbfejlesztési lehetőségek 44
8.1 Megvalósítás és tesztelés eredményei 44
8.2 Továbbfejlesztési lehetőségek 44
8.2.1 QoS szintek állíthatósága 44
8.2.2 További optimalizálás 44
8.2.3 UWP háttérmunkálatok 44
9 Tartalmi összefoglaló 45

10 Summary 46
11 Irodalomjegyzék 47
12 Ábrajegyzék 49
13 Táblázatjegyzék 50

1 Bevezetés

A technológiák sebes fejlődése, valamint a szoftveres újítások gyakran nyílt forrásúvá tétele szinte korlátlannak tűnő fejlesztési lehetőségekkel lát el minden mérnököt és tervezőt. Az első feladat az, hogy a megannyi lehetőség közül válasszák ki azokat, amiket meg akarnak valósítani. Ehhez ismerni kell az eszközt, amit fejleszteni akarnak legalább annyira, hogy meg tudjanak fogalmazni egy problémát, amire megéri megoldást találni. A probléma jellege utalhat például arra, hogy az eszköz képes lenne hatékonyabban működni, ha egy bizonyos funkcióját máshogy valósítanák meg, vagy akár arra, hogy az eszköz fejlesztése, esetleg átdolgozása valamilyen módon javíthatna a felhasználói élményen, ezzel felhasználóbarátabbá téve azt.

Két hallgatótársam és jómagam háromfős csapatként egy a felhasználói élmény javítását megcélzó rendszer tervezésébe fogtunk bele projektmunkánk gyanánt. A projekt célja egy távoli elérésű egyfázisú forgógép védőmodul és az ezt kezelő rendszer tervezése és elkészítése. Az egyfázisú forgógép elnevezés alatt olyan villamos motort érthetünk, amelynek az áramellátása egy vezetéken keresztül történik. Az adott motorra rászerelt védőmodul pedig a motor esetleges meghibásodásának elkerüléséért felel, ennek automatikus lekapcsolásával, ha például túl nagy feszültséget kapna a bemenetére. A rendszer lényegében a motorvédő modul köré épül fel, mivel a távoli elérés, vagy távkommunikáció megvalósítható egy ilyen védőmodullal. Ennek a kommunikációs rendszernek pedig az egyik eleme egy platformfüggetlen alkalmazás, aminek a segítségével a felhasználó bárhonnan el tudná érni a védőmodult. Én ennek az alkalmazásnak a tervezését és megvalósítását vállaltam el.

Az projektmunka során megoldani kívánt probléma az adott területen alkalmazott eszközök felhasználói élményével kapcsolatos, ugyanis az ehhez hasonló védőmoduloknál a távoli elérés általában az eszköz telefonos hívását, vagy erre való SMS (Short Message Service) küldését jelenti. Ezeknél az elérési módszereknél azonban már van lehetőség a felhasználó számára kényelmesebb módszer megvalósítására. Az IoT-t (Internet of Things) megvalósító eszközök és hálózatok rohamos fejlődésével ma már szinte bármilyen elektronikai eszközre találhatunk olyan megvalósítást, amit el tudunk érni bárhonnan egy mobilalkalmazással, amennyiben rendelkezünk internetkapcsolattal. Esetünkben a projekt tervezése során feldolgozott irodalom szerint a motorvédő modulok területén ez a fejlesztési lehetőség még nem lett megvalósítva, ezért gondoltuk úgy, hogy megpróbálnánk megtenni a lépést ebbe az irányba. Az általunk tervezett rendszer egy távoli elérésű egyfázisú motorvédő modulból, egy platformfüggetlen alkalmazásból, valamint egy szerverből állna, amin keresztül az alkalmazás és a modul kommunikálna. A rendszer kommunikációja az MQTT (Message Queuing Telemetry Transport) kommunikációs protokollt követné, ahol a protokollban definiált bróker szerepét a szerverünk, a kliensek szerepeit pedig a védőmodul és az alkalmazás töltenék be.

Szakdolgozatomként a mobil és asztali alkalmazás a tervezését és megvalósítását választottam. Célom egy platformfüggetlen alkalmazás készítése, aminek segítségével a felhasználó felvehetné, vagy törölhetné általa megfigyelni kívánt védőmodulokat. A modul érzékelői által mért aktuális adatokat pedig a program megjelenítené, valamint ezeket a beérkező adatokat naplózná, arra az eshetőségre, ha a felhasználó később kíváncsi lenne rá, hogy milyen állapotban volt a motor korábban. Ezen felül az alkalmazás segítségével le és fel tudná kapcsolni a motort, ha a körülmények megfelelők, valamint abban az esetben, ha a szoftver a háttérben működne, figyelné a bejövő, mért adatokat, és értesítést küldene, ha vészhelyzet lépett volna fel, tehát ha a védőmodul automatikusan lekapcsolta volna a motort, vagy ha az ilyenkor beérkezett adat átlépett volna egy beállított határértéket.

Egy ilyen funkciókkal ellátott alkalmazás érdemlegesen meg tudná könnyíteni az ehhez hasonló motorvédő modulok távolról történő elérését, vezérlését és a mért adatok leolvasását. Könnyedségéből adódóan pedig feltételezhetőleg gyakoribbá tenné a motor állapotának ellenőrzését. Egy ilyen eredményeket elérő rendszer reményeim szerint közelebb tudná hozni a felhasználót az általa használt modulhoz és az ellenőrzött motorhoz, tehát összesítve javítana a távoli elérésű védőmodulok jelenlegi megszokott felhasználói élményen, ezzel véleményem szerint létjogosultságot adva a hasonló módon működő rendszereknek ezen a területen.

2 A probléma megfogalmazása

Először célszerű az elkészítendő szoftvert megközelíteni felhasználói szempontból, ugyanis a fejlesztés célja egy olyan rendszer tervezése és megvalósítása, amely elegendő információval képes ellátni használóját a motor állapotával kapcsolatban, miközben a program használata gyorsan megérthető, felépítése pedig könnyen átlátható. Ehhez tisztázni kell már korán az alkalmazástól elvárt funkciók listáját, hiszen ezek a tervezés azon alappillérei, amik köré el lehet kezdeni felépíteni a rendszer kezdetleges tervét.

A fentebb emlegetett szempontoknak, valamint az alkalmazástól elvárt funkcióknak együttes ismerete meg tudja adni a kellő eszköztárat egy korai menürendszer kidolgozásához.

2.1 Program feladatai

2.1.1 Megfigyelt védőmodulok kezelése

Az alkalmazás tipikus felhasználójáról feltételezhetjük, hogy potenciálisan több motorvédő modullal is rendelkezik. Ennek tekintetében a szoftvernek támogatnia kéne több megfigyelendő modul tárolását, valamint egy felületet, ahol a program valamilyen módon listázza ezeket az eszközöket.

Ezeknek az eszközöknek a kezeléséhez továbbá hozzá tartozik a képesség új modulok felvételére, meglévő egyedek törlésére a gyűjteményből, valamint a lehetőség, hogy a felhasználó leállítsa egy vagy több általa kiválasztott modul megfigyelését, abban az esetben, ha esetleg épp ott tartózkodik a helyszínen és nincs szüksége az alkalmazás szolgáltatásaira.

2.1.2 Védőmodul adatainak leolvasása

A felhasználó által kiválasztott modul aktuális beérkező adatainak megjelenítése. Ez lényegében az alkalmazás fő feladata, éppen ezért fontos, hogy az a felület, amin a fő funkció van minél letisztultabb és gyorsabban feldolgozható legyen.

2.1.3 Távvezérlés

Valamilyen bemeneti kezelőszerv biztosítása a felhasználó számára, amivel ki és be tudja kapcsolni a motort.

Az alkalmazás tervezésekor az egyik fő szempont az egyszerű és felhasználóbarát kialakítás, ezért fontos, hogy a modul vezérlése is egyértelmű legyen, tehát a felhasználó számára a vezérlőszerv alatt érthetünk egy egyszerű kapcsolót, vagy gombot.

2.1.4 Adatok naplózása

Beérkező mért adatok naplózása, és lehetőség ezen adatoknak a beérkezés időpontja szerinti sorrendbeli megjelenítésére, arra az esetre, ha a felhasználó kíváncsi lenne, hogy milyen állapotokat mértek a moduljai korábban.

2.1.5 Felhasználó értesítése

A felhasználó értesítése abban az esetben, ha vészhelyzet lépne fel, de nincs épp használatban az alkalmazás.

Vészhelyzet alatt itt többféle eseményt lehet érteni. Az egyik lehetőség, ha egy mért tulajdonság értékei két határérték által közrezárt tartományon kívül esnének. Ezeket a megfigyelendő tulajdonságokat és a határértékeiket a felhasználó választhatná meg. A másik eshetőség pedig a motor automatikus lekapcsolása a védőmodul által, amennyiben túlfeszültség lépne fel a rendszerben. Ilyen vészhelyzetek esetében fontos, hogy a felhasználó tudomást szerezzen a fennálló problémáról, ugyanis egy értesítésekért felelős szolgáltatás nélkül a felhasználó csak azután tudhatná meg, hogy vészhelyzet lépett fel, hogy ő maga megnyitotta az alkalmazást, és végignézte egyenként a figyelt modulok állapotait.

A figyelni kívánt tulajdonságok és a határértékek beállíthatóságából kiindulva ez a funkció igen hasznos lehetne a vészhelyzetek kikerülésének szempontjából is. Tegyük fel, hogy olyan értékeket mér a modul, amelyek még nem jelentenek vészhelyzet esetében történő automatikus kikapcsolást, azonban ezen, a modul által biztonságosnak ítélt tartományon belül az értékeken nagy lengéseket lehet tapasztalni. Amennyiben a felhasználó tud erről a lengésről, dönthet úgy, hogy a problémát megelőzve ő maga kikapcsolja a motort a távvezérlő funkcióval.

2.2 Feladatok összegzése, és a megoldandó probléma

Az említett, alkalmazással szemben állított feladatteljesítési igényeknek, valamint a fejezet elején megfogalmazott tervezési elveknek az ismeretében kezdtem meg a menürendszer kezdetleges felépítésének átgondolását, amiben fokozatosan, lépésről lépésre mélyebbre haladva szűkülne a fókusz a modulok gyűjteményétől egészen egy specifikus tulajdonság beállításainak felületéig.

Eszerint a felhasználó először a modulok kezelési felületén találná magát. Itt a modulok kezelési feladatának leírásánál megfogalmazott lehetőségekkel tudna élni. Egy modul kiválasztása esetén, annak a saját menüjére kerülne a használó, ahonnan le lehetne olvasni a beérkező aktuális adatokat, tulajdonságok szerint elkülönítve, valamint itt lenne található a távvezérlési funkcióért felelős bemeneti szerv. Ezek után, ha a felhasználó kiválasztaná az egyik tulajdonságot, az alkalmazás egy új listafelületre vinné, ahol a naplózott adatok felsorolása történne, időpont szerinti sorrendben. Ezen felül lehetősége

lenne megnyitni az adott tulajdonság beállításait, ahol be vagy ki tudná kapcsolni ennek a tulajdonságnak a megfigyeltségét, valamint a határértékeket is be tudná állítani. Egy ilyen, vagy ehhez hasonló felépítésű menürendszerben való tájékozódás véleményem szerint eléggé intuitív lenne a célzott könnyed használat eléréséhez.

A szakdolgozat kidolgozása során történő tervezés és megvalósítás megoldandó problémája és célja egy olyan működő platformfüggetlen szoftver elkészítése, ami eleget tesz a megfogalmazott feladatteljesítési igényeknek, miközben megvalósításában követi az előző bekezdésben leírt kezdetleges menürendszer fókuszt szűkítő szerkezeti elvét.

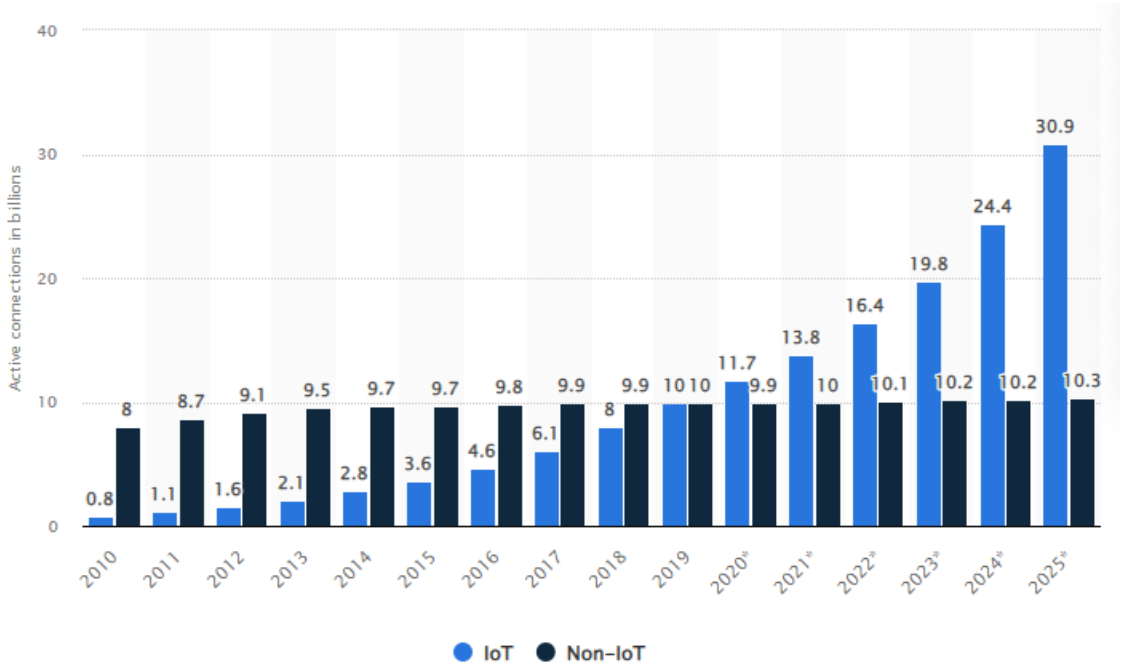
3 Irodalomkutatás

3.1 Internet of Things

Az Internet of Things, röviden IoT, olyan rendszereket ír le, amikben tárgyak vagy akár állatok és emberek is mind hálózatképes elektronikus eszközök használatával egymással kapcsolatban vannak az interneten vagy más hálózaton keresztül, és képesek az egymás közötti kommunikációra automatikus jelleggel. Az IoT definíciójára keresve az ember a korábbihoz hasonló leírásokból találhat végtelennek tűnő mennyiséget, de itt-ott kis eltéréseket találhat, attól függően, hogy ki fogalmazta meg, ugyanis nincs tisztázott definíció az Internet of Things-re.

A definíció tisztázatlansága rá is világít a terület jelenlegi talán legnagyobb, és az idő múlásával egyre fontosabbnak bizonyosuló kihívására, ami a szabványosítás. [1] Ezt a két megállapítást még 2015-ben fogalmazták meg a Jounal of Computer and Communications című folyóirat egyik számában kiadott cikkben, amire azóta már körülbelül 1080 alkalommal hivatkoztak különböző publikációkban az évek során. [2] Én az Internet of Things vizsgálatához ezt a tanulmányt vettem alapul, mert ebben az addigi irodalom feldolgozását, vizsgálatát és a terület véleményezését tették meg. Ebben a cikkben az írók megállapítják a szerintük legpontosabbnak vélt definícióját az IoT-nek, ami általam fordítva a következő: „Egy nyílt és széleskörű hálózata olyan intelligens tárgyaknak, amik képesek az automatikus szerveződésre, valamint információ, adat és erőforrások megosztására, mindezt úgy, hogy reagálnak és cselekszenek a különböző helyzetekkel és környezeti változásokkal találkozva.” [3]

Ezen felül a tanulmány azt is megállapítja, hogy a jobb szabályozhatóság érdekében a területnek sztenderd protokollok bevezetésére, a technológiák változatosságának fényében pedig jobb átjárhatóságra van szüksége az IoT rendszerekben résztvevő eszközöknek. [3] Annak ellenére, hogy ezek a törekvéseknek határozottan még nem valósultak meg [1], az IoT eszközök közötti aktív kapcsolatok száma közel megháromszorozta magát 2015-től 2021-ig, 3,6-ről több, mint 10 milliárdra, továbbá egy tavalyi előrejelzés – ami 2015 és 2019 közötti adatokra alapoz – 2025-re közel 31 milliárd aktív IoT-s eszközök közötti kapcsolat létét jövendöli, ami több mint háromszorosa a 2021-ben megállapított számnak. [4] [5] Ezen felsorolt adatok vizualizációját a 3.1. ábra szemlélteti. [4]



3.1. ábra. Aktív IoT és nem IoT kapcsolatok mennyisége az évek során

Ez egy olyan innovációnak vélhető, ami jelenleg úgy tűnik, hogy addig fog ilyen rohamosan terjeszkedni, amíg nem készül IoT-s alternatíva minden meglévő és új eszközre, amire elméletben készíthető ilyen elveken működő megvalósítás. A szakterületi normák megfogalmazására pedig érdemlegesen talán majd akkor kerülhet sor, ha a technológiákkal való kísérletezés előbb-utóbb alábbhagy, miután a cégek és fejlesztőik kiismerték melyek a leggazdaságosabb, legjobb teljesítményű, leghatékonyabb és ezek összesítésében legjobb technológiák és kommunikációs protokollok a szakterületen való fejlesztéshez. Az terjedő szabványok elismerése határozottan egyszerűbb feladat lesz, miután beálltak a terület normái és általánosan használt protokolljai, – ami egyébként az elmúlt pár évben már aktuálisnak kezd bizonyulni [6] – azonban ez nem azt jelenti, hogy nincsenek aktív próbálkozások a szabványosításra bizonyos szervezetek vagy nagycégek által is. [7]

Összesítésben kijelenthető, hogy a jóslatok az így működő eszközök elterjedéséről beigazolódtak, és a szakterület bebiztosította helyét a globális kommunikációs hálózatok világába, ezen túl pedig valószínűsíthető, hogy egészen addig fogunk hallani a terület potenciáljáról és sikereiről, amíg alapvető részévé nem válik a világ információs rendszerének.

3.1.1 Távoli elérésű motorvédő modulok állapota

Korábban említésre került, hogy a projekt készítése közben a feldolgozott irodalom alapján azt tudtuk megállapítani, hogy még ez a fajta fejlődési irány, amit az IoT nyújt, nem lett bejárva a dolgozatunkban leírt motorvédő modulhoz hasonló eszközök területén. Vannak példák olyan termékekre, amikkel lehet táv-kommunikálni hívással, SMS-sel

vagy akár mobilalkalmazással, azonban a hívásos és SMS-es kommunikációnál hozzáfűzendő, hogy ezek használatához SIM-kártyára (Subscriber Identity Module) van szüksége az eszköznek, valamint az alkalmazásos eléréshez egy Wi-Fi (Wireless Fidelity) hálózatra kell csatlakoznia a védőmodulnak és a mobilalkalmazásnak is. [8]

Az eszközök működésének vizsgálata során bizonyosodtunk meg arról, hogy megérné egy olyan alternatívának a kidolgozásába kezdeni, ami egy a területen újfajta, IoT alapú kommunikációs rendszert használ, ezzel hozzátéve a már emlegetett tendenciához, miszerint mindenféle eszközre, ami csak eszünkbe juthat előbb utóbb lesz IoT-s alternatíva is.

A rendszert három részre és szerepre lehet bontani. Elemeit tekintve egy vagy több motorvédő modulból, a szerverből és az alkalmazásból fog állni. Szerepek szempontjából pedig a modul célja a motor meghibásodásának megelőzése. Ezt a rákötött, motor tulajdonságait figyelő érzékelők által mért adatok feldolgozásával és ellenőrzésével fogja teljesíteni, majd ezeket az adatokat továbbítani fogja, hogy bárhonnan leolvashatóak legyenek. A szerver lesz az adatok továbbítója, és a távoli elérést megvalósító kapcsolat fenntartója a modul és az alkalmazás között. Végül pedig az alkalmazás fog nyújtani a felhasználó számára egy internetkapcsolat segítségével elérhető virtuális érintkező felületet a védőmodullal, amin keresztül látni fogja a motor legutóbb mért adatait, képes lesz vezérelni az eszközt, valamint hozzáférése lesz a probléma megfogalmazásáról szóló fejezetben leírt többi funkcióhoz is. A tervezett rendszer protokollt követő kommunikációs felépítése a következő alfejezetben olvasható.

3.2 MQTT lapú kommunikáció

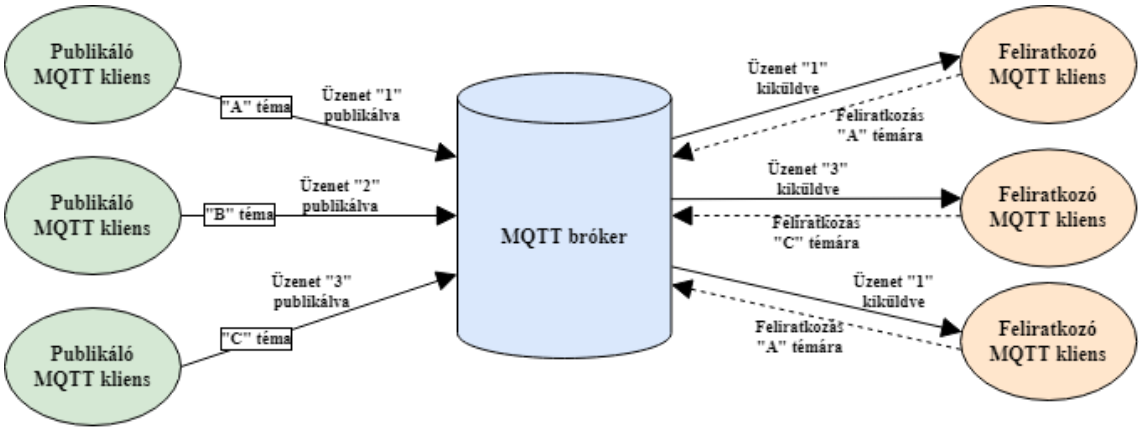
Az MQTT, vagy teljes nevén Message Queuing Telemetry Transport az egyike a korábban emlegetett IoT szabványosítási törekvéseknek, ráadásul eléggé sikeresnek tekinthető, ugyanis ez az egyik leggyakrabban használt protokoll, ha IoT kommunikációról beszélünk. [6] Megalkotásakor a fő szempont egy olyan kommunikációs protokoll meghatározása volt, amellyel pontosan lehet adatot küldeni nagy késleltetéses és kis sávszélességű hálózatokban. [9] Ezek a tulajdonságai egybeesnek az IoT-s kommunikációs hálózatok igényeivel, ezzel igen jó választásnak bizonyosulva. A protokollal először 1999-ben Andy Stanford-Clark és Arlen Nipper állt elő, ma pedig a szabványt az OASIS nonprofit konzorcium fejleszti, IoT-s alkalmazásra szánva. Az IoT kommunikáció jövőjét a szervezet az MQTT protokollban látja, ezt sugallja az is, hogy a protokoll weboldalán ez van megnevezve a terület definitív szabványaként, ráadásul ez az állítás az első dolog, amivel az oldal látogatásakor találkozhatunk. [10]

3.2.1 Működése

Habár az elnevezésben szerepel a „message queuing”, működését tekintve a protokoll nem az üzenetsoros mintára épít. Valójában a publish-subscribe, azaz a publikáló-feliratkozó minta az alapja. MQTT esetében az üzenet küldője, vagyis a publikáló nem közvetlenül egy specifikus címzettel folytat kommunikációt, hanem az üzeneteit bizonyos témákra, vagy topic-okra küldi a szerver felé. Miután megkapta az üzenetet, a szerver megnézi a megjelölt témát, és továbbküldi minden erre a témára éppen feliratkozott szereplőnek.

Az üzenetek beérkezésének biztosításáért három minőségi szintet különböztet meg a protokoll üzenetküldés szempontjából. Ezeket nevezzük „Quality of Service”, röviden QoS szinteknek. A szintek a QoS 0, 1 és 2 neveket viselik. A 0-ás szinten az üzenet kiküldése legfeljebb egyszer történik meg. Itt nincs biztosíték arról, hogy az be is fog érkezni. Az 1-es szint a legalább egyszeri kiküldés elve szerint működik, ahol a küldés egyszeri végrehajtása biztosított, de egy duplikáló jelző beállításával el lehet érni a kiküldés többszörösítését is. A 2-es szinten pedig a rendszer négyutas kézfogást használ. Ezt azért teszi, hogy a címzett egyszer biztosan, de többször ne kapja meg az üzenetet. Ezen szintek közül a 2-es a legbiztosabb, azonban ez lassabb kézbesítést is jelent. [9]

A protokoll egy tipikus MQTT hálózatban két féle szereplőt használ, ezek a kliens és a bróker. A szerver tölti be a bróker szerepét, minden elküldött üzenetet először ez kap meg, és továbbít minden vele aktuálisan kapcsolatban lévő olyan kliensnek, amely fel van iratkozva az üzenetben megjelölt témára. Másik oldalt a kliensek pedig működhetnek publikálóként, feliratkozóként, vagy egyszerre mindkettőként. Egy ilyen felépítésű MQTT hálózatot mutat be a 3.2. ábra.



3.2. ábra. Tipikus felépítése egy MQTT hálózatnak

3.2.2 MQTT a rendszerben

A távoli elérésű motorvédő modulhoz és az azt kezelő rendszerhez az MQTT kommunikációs protokoll lett választva. A védőmodul, valamint az általam készített platformfüggetlen alkalmazás a kliensi szerepet fogják betölteni, míg a szerver a bróker

lesz természetesen. A fejlesztés során a védőmodul helyettesítését egy általam készített, egyszerű konzolos alkalmazás fogja átvenni, ami úgy fog adatokat generálni, és azokra a témákra fogja ezeket publikálni, ahogyan ezt egy tipikus védőmodultól elvárnánk. Erre a helyettesítésre szükség lesz, mert a védőmodul nem lesz mindig bekapcsolva, vagy működőképes a fejlesztés és tesztelés alatt.

Kommunikációs feladatait tekintve az általam fejlesztett alkalmazás kliensként fel fog iratkozni a megfigyelt modulok, azon belül pedig a tulajdonságaik számára egyenként kijelölt témákra. Egy ilyen MQTT téma a következő mintát fogja követni: „modulazonosító/tulajdonság”. Az alkalmazás a beérkező üzeneteket a megjelölt témája szerint fogja elkülöníteni, és megjeleníteni a mindenkori legutóbb beérkezett értéket az adott modul adatleolvasásra biztosított felületén. Ezen felül a korábban mért adatok visszanézhetőségi elvárásának fényében a beérkezett adatot naplózza is. A távvezérlési funkció érdekében publikálni is fog a program. Az erre kijelölt témára akkor küldene üzenetet, ha a felhasználó élne a motor le-, vagy felkapcsolásának lehetőségével az adatleolvasó felületen lévő kezelő szervet használva. A modulok kezelésének szempontjából, abban az esetben, ha a felhasználó leállítaná egy modul tulajdonságainak figyeltségeit, akkor az alkalmazás leiratkozna és a megfigyelés visszakapcsolásáig nem iratkozna fel az adott témákra, így nem kapná meg a védőmodul által publikált adatokat. Abban az esetben pedig, ha a felhasználó felvenne egy új védőmodult, a megadott adatok alapján az alkalmazás legenerálná az új modul témáit, és feliratkozna rájuk.

3.3 Platformfüggetlen alkalmazás fejlesztés

Ahogy az igényspecifikációban is olvasható, a szakdolgozatban leírt szoftvernek a célja a motor állapotát úgy bemutatni kellő részletességgel, hogy mindeközben ez egyszerű, könnyen érthető és használható legyen. Röviden tehát legyen informatív, de kényelmes a használata. A fejlesztés gyakorlati részéhez használt keretrendszert és eszközöket ennek fényében érdemes megválasztani, mert ha az ember csak egy szempont alapján választ, legyen az megszokás vagy a keretrendszer népszerűsége, akkor könnyen lehet, hogy megnehezíti a saját dolgát, ugyanis valószínű, hogy léteznek olyan alternatívák, amelyek eszközei jobban igazodnak a fejlesztés céljaihoz.

Mivel a létrehozandó alkalmazás a kényelmet és a könnyű használhatóságot venné fókuszba, a program fejlesztéséhez használt módszerek közötti latolgatásnál már korán felmerült bennem a platformfüggetlen fejlesztés lehetősége. Ezen fejlesztési terület megismeréséhez egy 2021-ben íródott tanulmányt vettem alapul, amely során az írók gyakorlatban, ipari környezetben dolgozó fejlesztőkkel készített interjúk alapján értékelték ki a platformfüggetlen fejlesztés módszereit, valamint előnyeit és hátrányait. [11]

3.3.1 Célja

Alapvetően a több platformra történő fejlesztést úgy kellett elképzelni, hogy minden platformra, amire az alkalmazást ki akarták adni saját, új alkalmazást kellett készíteni, az adott platform gyakran sajátos kedvelt eszközeivel és programozási nyelveivel. Ebben az esetben a kész alkalmazások közös tulajdonságai a célfunkcióik, valamint esetleg a felhasználói felületük kinézete. Egyszerre több különálló alkalmazás fejlesztéséhez egy cégnek pedig valószínűleg szüksége van minden platformon egy saját csapatra. Az okostelefonok rohamos elterjedésével pedig a mobilalkalmazások használata sokak számára a mindennapi életük alapvető részévé vált, ezzel a mobilalkalmazás fejlesztést az egyik legnövekvőbb informatikai területté téve. [11] Egy ekkora, a 2010-es évek első felében még újnak számító területen sok cég és csapat azon lehető legegyszerűbb és legolcsóbb megoldások keresésébe fogott, amik a mobilfejlesztés fennálló problémáira adhattak megoldást. A platformfüggetlen fejlesztés ezeket az igényeket igyekszik kielégíteni úgy, hogy egy közös kódbázis megírását várják el a fejlesztőktől, amíg a platformfüggetlenséget megvalósító fejlesztői eszközök ezt a kódot az adott célplatformokon futtatható kódokra fordítják, így használhatóvá téve az adott keretrendszer által megjelölt mindegyik platformon az egyszer megírt szoftvert.

3.3.2 Előnyei, hátrányai

Az így történő fejlesztés legegyértelműbb előnye a céljából adódik, ami az, hogy lehetőség szerint elég egy közös kódbázis mindegyik megjelölt platformra. Ennek az alapvetésnek a következménye a fejlesztési módszer minden előnye, beleértve a hatékonyabb fejlesztést és gyorsabb prototípus készítést, valamint azt a tényt, hogy a készülő program minden platformon ugyanott tart a fejlesztésben, ezáltal mire minden platformspecifikus hiba ki lett javítva, készen áll az összes platformon az alkalmazás. A hatékonyságot tovább ecsetelve pedig még megemlítendő, hogy az ily módon készített programok fejlesztésébe fektetett munka és a kellő jártasság a célplatformonként különböző rendszerek natív eszközeiben töredékére esik, így imponálóbb opcióvá téve ezt az alacsony költségvetésű, esetleg kis létszámú fejlesztőcsapatok számára. [11]

A módszer hátrányai is természetesen az alapvetéséből adódnak. Kódot írni egyszerre több nagyban különböző platformra, amiknek a natív eszközei is gyakran egymástól eltérően működnek nehéz feladat, de a platformfüggetlen fejlesztési eszközök használatánál ez nem a szoftver fejlesztőjén csapódik le, hanem az adott keretrendszer vállalja át. Ezen felelősség átvállalásának nehézségei azonban az idő múltával, a keretrendszereknek és fordítóknak a folyamatos fejlődésével és okosodásával, ha lassan is, de oldódnak. Ezzel szemben annak ellenére, hogy ezek a keretrendszerek okosodnak, a különféle platformok működései és eszköztárai is folyamatos változásnak és fejlődésnek vannak kitéve, amit ezeknek a keretrendszereknek követniük kell, és alkalmazkodni hozzájuk, ezzel is lassítva az okosodásukat. Érdemes még arra is kitérni, hogy a korábban említett tanulmányban a megkérdezett fejlesztők, akik többféle cégnél

dolgoztak a beszélgetés idejében a következőkben leírt ellenérveket tudták megállapítani a platformfüggetlen fejlesztéssel szemben, tapasztalataik alapján. Ha valós idejű, teljesítményigényes alkalmazások fejlesztése a cél, mint például játékok, akkor a natív eszközökkel való fejlesztés a javasolt, valamint abban az esetben, ha sok natív eszköz van használva egy platformfüggetlen alkalmazásban, akkor a szoftver támogatásának időigénye és a natív eszközökhöz való hozzáértés igénye hasonlóvá válik egy natív eszközökkel késztett alkalmazáséhoz. [11]

Alkalmazás készítése platformfüggetlen fejlesztői eszközökkel	
Előnyök	**Hátrányok**
Egy kódbázis több platformra.	Kész szoftver teljesítménye rosszabb egy natívan írt alkalmazáséhoz képest.
Gyorsabb prototípus készítés.	A keretrendszernek le kell követnie a célplatform változásait, ez késve történik.
Mindegyik célplatformon ugyanott tart a fejlesztés.	Sok natív eszköz használata esetén a szoftver támogatása olyan, mintha natív lenne.
Kevesebb kellő jártasság különböző rendszerekben.	
Hatékonyabb fejlesztés.	

3.1. táblázat. Platformfüggetlen fejlesztés előnyei és hátrányai

Ezen előnyök, hátrányok (amelyek a 3.1. táblázatban rendszerezetten is olvashatóak) és irodalom figyelembevételével maradtam azon az álláspontón, hogy az általam készített alkalmazás fejlesztését platformfüggetlen módon fogom végezni. Miután a döntésben sikerült megbizonyosodni, elkezdtem a rendelkezésre álló keretrendszerek közül válogatni és utána járni, hogy melyek azok, amelyek a számomra legelőnyösebbek lehetnek. Mivel eléggé sok ilyen fejlesztői rendszer létezik, ezért a vizsgálathoz két keretrendszerre szűkítettem a fókuszt, a Xamarinra és a React Native-re, előbbit a keretrendszerrel használható eszközökkel való jártasságomnak, utóbbit pedig a területen való népszerűségének fényében.

3.3.3 Xamarin

A Xamarin egy olyan nyílt forrású, Microsoft által támogatott fejlesztői eszközök összessége, ami a .NET keretrendszer eszközeinek kiterjesztését teszi lehetővé platformfüggetlen, C# nyelven íródó alkalmazások készítéséhez. A rendszer támogatott platformjai az iOS, Android és UWP (Universal Windows Platform). Eszerint a rendszerrel egyszerre lehet a kétféle domináns mobilplatformra, valamint minden Windows-t használó eszközre fejleszteni egy kódbázissal. [12]

A Microsoft leírása szerint a fejlesztők olyan alkalmazások fejlesztésére képesek a rendszerrel, amikben a platformok között megosztható kód mennyisége magasabb, mint más keretrendszereknél. A kevesebb platformspecifikus kód ígérete mögött az rejlik, hogy a keretrendszert arra lehet használni, hogy az alkalmazásoknak egy közös C# logikát, adatbáziskezelési kódot és hálózati kommunikációt lehet írni. Az imponáló marketingtől és ígéretektől függetlenül viszont abban az esetben, ha az egyik platform újításokkal áll elő, vagy a már régóta valamilyen módon működő rendszerükben változtatnak, akkor a keretrendszernek ezeket a változásokat le kell követnie, és alkalmazkodnia kell az új működéshez, hogy valóban natív érzésű alkalmazásokkal tudjon előállni. Ennek eredményeképp az újítások viszont később érik el a Xamarinnal dolgozó fejlesztőket gyakorlatban, mert a keretrendszer változások esetén garantáltan le lesz maradva. [12] [13]

A felhasználói felület elkészítéséhez a XAML (Extensible Application Markup Language) nyelv használható. Működését tekintve a keretrendszer az ebben a leíró nyelvben meghatározott felhasználói felület komponenseinek platformspecifikus megfelelőit használja a különböző platformokra fordított alkalmazásokban. A nagymennyiségű osztott kódon túl a Xamarinnal készített alkalmazások mellet még nagy érv ezen alkalmazásoknak a teljesítménye, ugyanis a közös C# kód mind iOS-en, mind Androidon a sajátos natív assembly kódra fordul. [12] [13]

3.3.4 React Native

Ennek a keretrendszernek a fejlesztését a Facebook (azóta már Meta) indította az addigi React nevű, JavaScript nyelvű felhasználói felületek fejlesztésére szánt osztály könyvtárukra építve 2015-ben. Eddigi hét éves pályafutása alatt az egyik legnépszerűbb platformfüggetlen keretrendszerré vált. [14] A keretrendszerrel készíthető többek között Android, iOS és Windows platformokra is szoftver. Egy alkalmazás megvalósítása során a fejlesztőnek lehetősége van JavaScript-ben írnia a készülő alkalmazást, ami lefordulva pedig natív eszközöket használ a célplatformokon, ezen túl pedig a rendszer lehetőséget biztosít arra is, hogy a készítő írhasson platformspecifikus natív kódot, Android esetében Java vagy Kotlin, iOS esetében Objective-C vagy Swift és Windows platformra pedig C++ vagy C# nyelveken, ezzel igen széleskörű eszköztárat biztosítva a fejlesztői csapatok számára. Mivel az eredetileg webes felületek fejlesztésére készített React könyvtárra épít a keretrendszer, valamint lényegében ezen könyvtárnak a használhatóságának kiterjesztését szolgálja, azok a fejlesztők, akik webalkalmazások fejlesztéséhez szoktak jól tudják hasznosítani tapasztalatukat a JavaScript-tel való programozás során. [15]

A platformfüggetlen fejlesztés általános előnyeivel és hátrányaival a React Native is rendelkezik, ilyenek például a fejlesztés időtartamának és a költségeinek töredékére esése a natív fejlesztői csapatok fejlesztési idejeivel és költségeivel szemben, vagy ilyen még a karbantartás leegyszerűsödése, mivel egy kódbázisban lehet megoldani a felmerülő problémákat. Hátrányok szempontjából pedig a teljesítményproblémákat lehetne

felhozni, általánosságban nem is tanácsos platformfüggetlen keretrendszerekkel magas erőforrás igényű alkalmazásokat készíteni. A platformfüggetlen fejlesztés általános tulajdonságain túlmutatva viszont a React Native mellett elég erős érvnek bizonyul a rendszer Fast Refresh (gyors újratöltés) funkciója, aminek köszönhetően a kódban történt változtatások akár egy-két másodperc alatt láthatóak a virtuális eszközön, ahol le sem kellett állítani az alkalmazás futását. [16] Ez szintén olyan előny, aminek köszönhetően nagy időt tud megspórolni az adott csapat a fejlesztés során.

3.4 Értesítési rendszerek

Értesítési rendszerek tekintetében is olyan megvalósítási módszerek elemzése célszerű, amelyeknek a működési alapvetéseik összefüggenek a készítendő rendszer céljaival. Ebből kiindulva úgy döntöttem, hogy a tárgyalt értesítési lehetőségek az e-mail, valamint a lokális (local) és távoli (remote) leküldéses értesítés (push notification) küldési funkciók legyenek. A kihagyott lehetőségek között van az SMS-t, valamint felhasználó hívását használó értesítési funkciók. Ezek a megvalósítási lehetőségek azért nem kerülnek bemutatásra és elemzésre, mert működésünkben közelebb állnak a már létező távvezérlésű motorvédő modul rendszerekhez, mint ahhoz, aminek a része az általam készített alkalmazás. A piacon található távvezérlésű modulok gyakran SMS-en vagy híváson keresztül történő távoli elérést valósítanak meg. [8] Emiatt egy eseményhez kötött SMS küldése vagy hívás indítása lenne az egyértelmű fejlesztési lépés, ha ezen készülékek tervezői úgy gondolnák, hogy szeretnének egy értesítési funkciót megvalósítani. A készülő rendszer egyik célja alternatívát adni a már létezőkre, és habár nem tudom, hogy terveznek-e cégek egyáltalán hasonló funkciót megvalósítani, úgy gondoltam, hogy kihagyom azokat a megvalósítási módokat, amelyek irányába valószínűleg elindulnának a már létező rendszerek.

3.4.1 E-mail

Egy rendszer felhasználóinak automatikusan generált e-mailek küldésén keresztül történő értesítése régóta bevett gyakorlat, és a hatásosságát jelzi, hogy a mai napig milyen sokféle területen alkalmaznak e-mailes értesítési funkciót. Ez lehet gyakori vagy ritka rendszerességű hírlevél generálása és küldése nagycégek által belső kommunikációs rendszereikben, lehet egy újság kiadványának megjelenésével kapcsolatos értesítő e-mail, vagy akár arról is kaphatunk küldeményt, ha valaki egy kommentet fűzött egy bejegyzésünkhöz az egyik általunk használt közösségi média felületen. Az online térben gyakorlatilag bármely SMTP (Simple Mail Transfer Protocol) szerverrel rendelkező rendszer bármely eseményéhez lehetséges kötni egy e-mailes értesítési funkciót. Ennek a megvalósítása pedig az idő múltával és a technológia fejlődésével egyre egyszerűbbé válik a kis költségvetésű, vagy akár hobbiból tervezett rendszereket kivitelező csapatok és személyek számára is, mivel a nyílt forrású modellt követő szoftverek elterjedése és folyamatos fejlődése igen választékos eszköztárat ad a tervezők kezébe, például ebben az

esetben arra, hogy elkészíthessék a saját SMTP szerverüket. Azonban egyszerűbb eseteknél, ahol nincs szükség a nagy skálázhatóságra, még erre sincs szükség gyakran, hiszen vannak ingyenesen használható SMTP szolgáltatások.

3.4.2 Push értesítés, lokális és távoli

Felhasználói szempontból a push értesítés egy olyan felugró üzenet az adott készüléken, amit kiválasztva a rendszer általában az értesítés szövegéhez releváns menüjére juttatja a használót. Ez a fajta értesítési megoldás kényelmes a felhasználók számára és a megjelenése óta annyira elterjedt a használata, hogy manapság az Amerikai Egyesült Államokban az átlagos okostelefont használó egyén napi 46 push értesítéssel találkozik. [17] Technikai szempontból egy ilyen szolgáltatást az adott rendszeren kétféleképpen lehet megvalósítani. Ezek a megvalósítási típusok a lokális és a távoli módszerek. Elnevezésük beszédes, a lokális push értesítéseket a célzott mobil indítja és kapja is, a távoli push értesítések esetében pedig egy szerver küld le egy eseményt az interneten keresztül minden címzett készülék számára.

A lokális megoldás implementációja úgy működik, hogy az értesítést az adott alkalmazás építi össze, majd állítja be, hogy mikor dobja fel az okostelefon. Ezt a módszert tipikusan időhöz kötött emlékeztetésekhez használják, például, ha a felhasználó beállított magának egy naptári eseményt, ami ha közeledik, a rendszer értesítést küld, ezzel sikeresen emlékeztetve a felhasználót. [18]

Ezzel szemben a távoli megoldás a már korábban ismertetett publikáló-feliratkozó modellt követi, és az értesítés felugrásáért egy erre a feladatra kijelölt szerver által leküldött esemény felel, amiknek a leküldése a feliratkozott kliensek számára az esemény bekövetkezésével elméletben közel egyszerre történik meg. [19] Gyakorlatban azonban az ezt a megoldást megvalósító rendszereknek szembe kell nézniük azzal a problémával, hogy nem mindenhol és mindig kapcsolódnak a célzott készülékek az internethez. Ezt úgy oldja meg a módszer, hogy az eseményt küldő szerver egy bizonyos ideig tárolja az értesítési üzenetet, és ha a célkészülék ezen az időintervallumon belül újra felcsatlakozik az internetre és kapcsolatba lép vele, akkor a szerver ennek a címzettnek is leküldi az eseményt. [20] Használatát tekintve a távoli push értesítéseket általában olyan rendszereknél használják, ahol egy szerver tud arról, ha egy fontos esemény bekövetkezett, amiről célszerű lenne értesíteni az adott alkalmazást használó készülékeket.

3.5 A megoldási módszerek kiválasztása, ezek indoklása

A vizsgált rendszereket és megvalósítási lehetőségeket elemezve, átvizsgálva és értelmezve a következőkben felsorolt rendszereket és módszereket választottam a szakdolgozatban leírt szoftver elkészítéséhez.

3.5.1 Xamarin

A platformfüggetlen fejlesztést lehetővé tévő keretrendszerek közül a Xamarinra esett a választásom több szempontot figyelembe véve is. Ezzel a keretrendszerrel C#-ban lesz lehetőségem alkalmazások készítésére a .NET fejlesztői eszközeivel, a felhasználói felületet pedig XAML nyelvben lesz lehetőségem meghatározni. Ezek mind olyan eszközök és nyelvek, amikkel az egyetemi képzésem során több félév és tárgy keretében készíthettem alkalmazásokat. Mivel mobilalkalmazást még nem fejlesztettem, az a szintű jártasság az ehhez használt fejlesztői eszközökben, amivel rendelkezem biztosan előnyömre fog válni.

Végül pedig egy 2020-ban íródott folyóiratcikk adatösszehasonlításainak tanulmányozásába fogtam, hogy ne csak személyes vélemény alapján döntsek. Ebben a cikkben négyféle platformfüggetlen keretrendszerben megírt, azonos feladatot ellátó, Android rendszeren futó tesztalkalmazás következő tulajdonságait hasonlították össze: a szoftver installációs fájljának mérete, futtatás során az alkalmazás által létrehozott forrás fájlok mérete, renderelési vagy kirajzolási ideje, futásidőben a processzor kihasználtsága, az alkalmazás által használt memória mennyisége és az alkalmazás fogyasztása. A négy tesztelt keretrendszer között szerepelt többek között a React Native és a Xamarin is. A tesztalkalmazás feladata pedig egy 1000 elemű, stílusokkal ellátott lista kirajzolása. Az emlegetett tesztesetekben a React Native-vel készített alkalmazás installáló fájlja volt a második legnagyobb méretű, létrehozott forrás fájlok méretét tekintve a legnagyobbnak, kirajzolási időben a leglassabbnak, processzor kihasználásában a legnagyobbnak, az alkalmazás által felhasznált memória tekintetében a legnagyobbnak, és energiafelhasználásban is a legnagyobbnak bizonyult. Ezzel szemben a Xamarinnal készített tesztalkalmazás installáló fájlja volt a legkisebb méretű, a létrehozott forrás fájlok méretét tekintve a legkisebbnek, kirajzolási időben a React Native-os alkalmazás után a második leglassabbnak, processzor kihasználtságában a legkisebbnek, felhasznált memória tekintetében a legkisebbnek és energiafelhasználásban pedig átlagosan a legalacsonyabbnak bizonyult. [14]

Az ezen tesztesetek során mért adatok alapján ezeket a következtetéseket vontam le: a Xamarinnal készített alkalmazások a többi vizsgált, népszerű keretrendszerekkel készített alkalmazásokhoz képest kevesebb területet foglalnak le a futtató eszköz háttértárán, futtatás során pedig a processzorkihasználtság tekintetében stabilabb és alacsonyabb értékeket produkálnak, a rendelkezésre álló memóriából is kevesebbet foglalnak le, valamint energiafelhasználási szempontból is stabilabbnak, és átlagosan alacsonyabb fogyasztásúnak bizonyultak. Ezeket összesítve kijelenthető, hogy a Xamarinnal készített alkalmazások kevesebb erőforrást használnak fel, ezzel könnyebbé téve használatukat a hardver számára. Ezen hatékonyságnak azonban az a hátulütője, hogy a kirajzolási ideje eléggé le volt maradva a legjobb értéktől, de a tervezett alkalmazás céljait tekintve ezek az értékek szintén arra utalnak, hogy ez lenne a megfelelő keretrendszer.

3.5.2 Lokális push értesítések

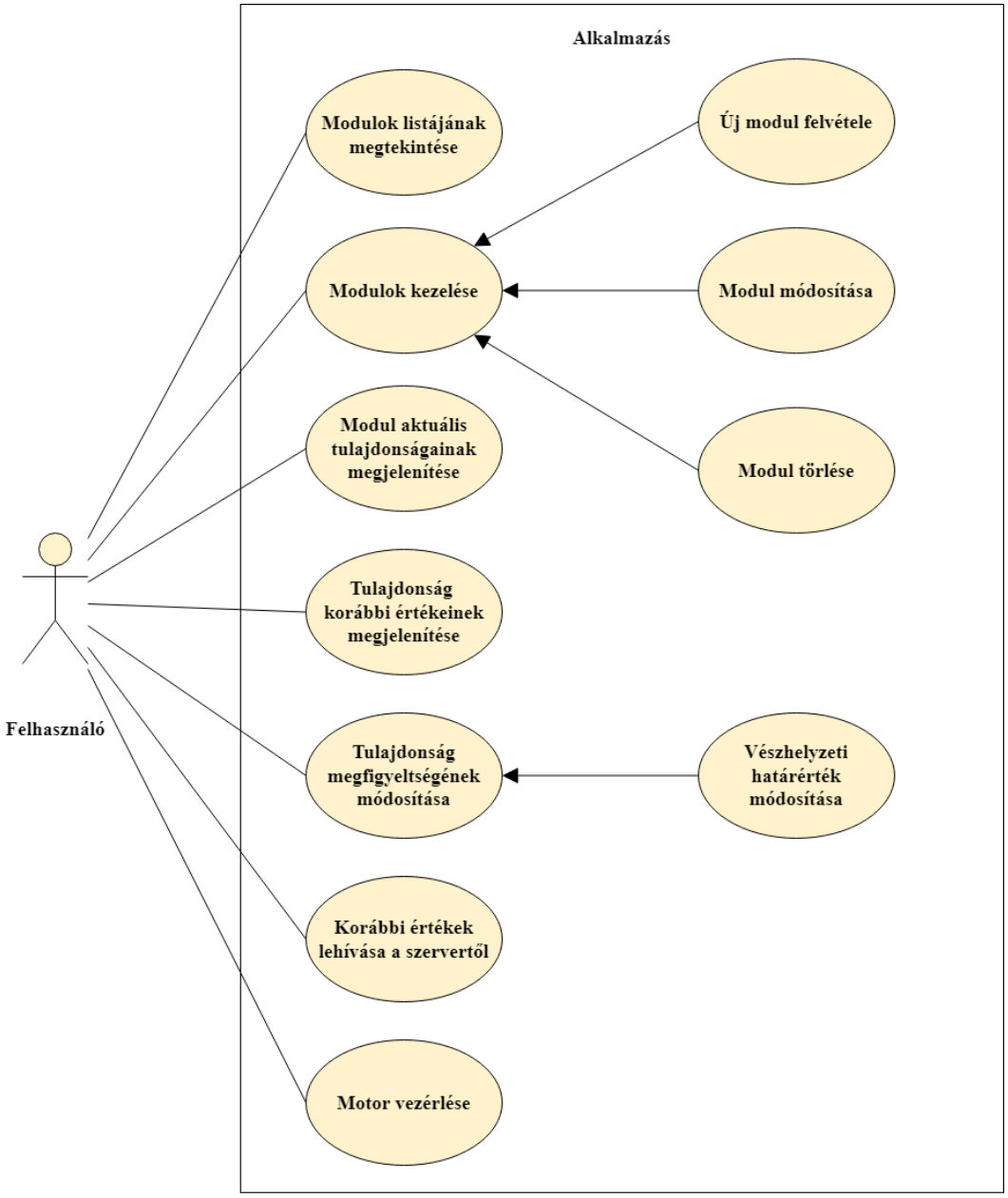
A felhasználó értesítésének szempontjából a lokális értesítések megvalósítását választottam. Azért esett erre a lehetőségre a választásom, mert egy alkalmazásban akartam megoldani a többi elvárt funkcióval együtt ezt is. Az e-mailek küldéséhez, valamint a távoli push értesítések használatához mindkét esetben külső hardveres és szoftveres erőforrások használatára lenne szükség, ami az e-mail küldözgetés esetében az én általam kívánt funkció delegálása lenne a védőmodult kezelő rendszer brókerének irányába, aminek fejlesztésével azonban nem én foglalkozom. Távoli push értesítések esetében pedig egy külső szerveres szolgáltatásra lenne szükség, aminek összeköttetésben kellene állnia a brókerrel, hogy tudja, milyen esetben és melyik felhasználónak kellene értesítést küldenie. Ez véleményem szerint feleslegesen bonyolítaná tovább a rendszer felépítését és működését, amikor létezik egy olyan alternatíva, amivel ezt a feladatot helyben, a mobilalkalmazáson belül meg lehet oldani.

4 Igényspecifikáció

Az eddigiek során megfogalmazott fejlesztési célok, elvek, felvetett feladatteljesítési igények, valamint kiválasztott megoldási módszerek tudatában kezdtem el a rendszer megtervezését. A rendszernek ezen, igények alapján felállított tervét lehet a következőben olvasni.

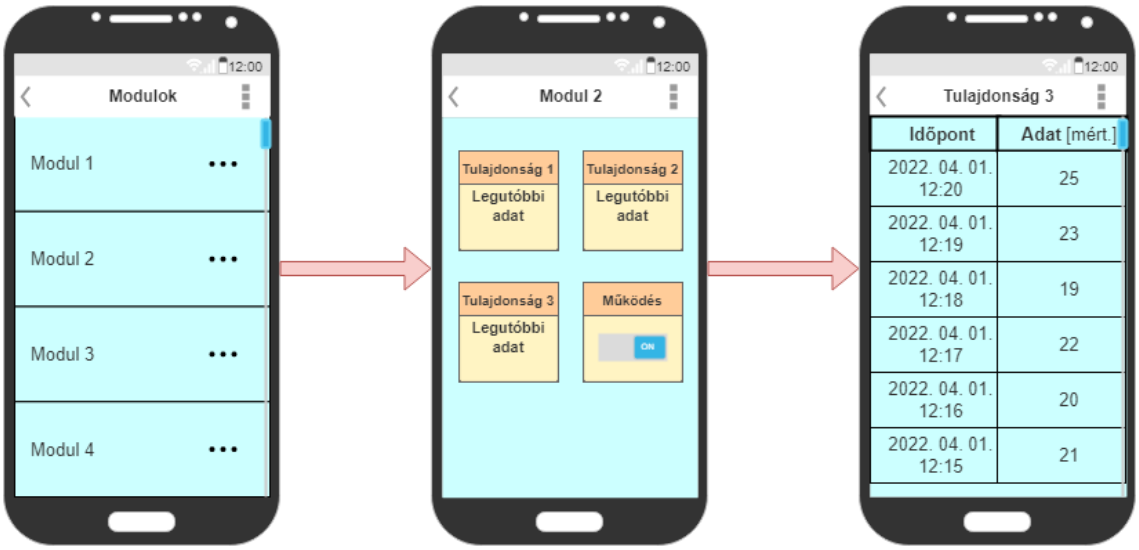
4.1 Funkciók és menürendszer

A probléma megfogalmazását leíró fejezetben felsorolt elvárások alapján az alkalmazás használati eset diagramja a 4.1. ábrán látható módon épül fel.



4.1. ábra. Az alkalmazás használati eset diagramja

Az ábrán látható használati esetek és a kezdetleges menürendszer fókuszt szűkítő alapelvének fényében a felhasználói felület menürendszerének terve a következő.



4.2. ábra. Az alkalmazás tervezett menürendszere

A program futtatásakor a 4.2. ábra első menüjén, a tárolt modulok listájával találkozik a felhasználó, itt a modulok kezelésével tud foglalkozni. A jobb felső sarokban található gombot lenyomva választhatja ki az új modul felvételének funkcióját, a modulok módosításának, vagy törlésének funkcióját pedig az adott modul elemhez tartozó kontextus menü megnyitásával képes elérni. A felhasználó egy modul elemet kiválasztva az adott egyed aktuális tulajdonságainak megjelenítését hívja meg, aminek csempézetten kialakított elrendezését az ábrán a második menü jelzi. Itt megtalálható a legutóbbi adatok megtekintésének funkcióján kívül a motor vezérlésére használt kapcsoló is, ami az ábrán egy egyszerű kétállapotú kapcsolóként van feltüntetve, azonban működését tekintve komplexebb lesz az egyszerű, utoljára felhasználó által beállított érték megjelenítésénél, mert nem elég azt tudni, hogy a felhasználó utoljára milyen állapotba állította a motort, hanem azt is meg kell jeleníteni, hogy a kapcsoló állapotától függetlenül a motor működésben van-e. Ennek a szükségessége abban rejlik, hogy a felhasználónak rögtön kellene látnia, ha a vezérléssel probléma akadt, mert a motor esetleg továbbra is problémamentesen működik, annak ellenére, hogy ő megpróbálta lekapcsolni korábban. Ezen a menün továbbá a jobb felső gombra nyomva tudja a felhasználó lekéretni a szervertől a tulajdonságok korábbi értékeit. A csempézett menün egy tulajdonságot kiválasztva egy újabb listafelülettel találkozik a felhasználó, ahol az adott tulajdonság korábban beérkezett értékeit jeleníti meg, idő szerinti sorrendben. A felhasználó ezen a felületen végül pedig a jobb felső gombot lenyomva az adott tulajdonság háttérben való megfigyeltségének módosításával képes élni, valamint a tulajdonság vészhelyzeti határértékeinek beállítását is itt lehet megtenni.

A leírt felépítés alapján megvalósított menürendszer követi a fókusz szűkülésének elvét, valamint a használati esetek diagramján meghatározott feladatokat és a fejlesztési célokat

is teljesíti. Ezek alapján pedig véleményem szerint az elvárásokat kielégítő felhasználói felületet biztosít az alkalmazásnak.

4.2 Adatok tárolása

Az alkalmazás gondolati síkon történő létrejöttének idejében még kérdéses volt számomra, hogy az adatok tárolását miként kellene megoldani. Először, kevés adatra számítva, szövegfájlokban tárolt modul azonosítókban gondolkodtam, azonban ahogy kezdett kirajzolódni, hogy milyen egyenként eltérő és közös tulajdonságokkal kell bírnia a védőmoduloknak, valamint hogy a közös tulajdonságok értékeit érdemes megőrizni a felhasználó számára, egyre egyértelműbbé vált, hogy egy helyi adatbázis használatára lesz szükség. Ennek az adatbázisnak a tábláiban lesznek eltárolva sajátos jellemzőkkel, a beérkező mérési értékek, beérkezésük időpontjaikkal, a modulok, valamint a modulok tulajdonságai, hogy el lehessen tárolni, melyek lesznek megfigyelve.

4.3 Kommunikáció

Mint az már ismertetésre került, a rendszer kommunikációja az MQTT protokoll szerint fog működni, ebben a kommunikációs hálózatban pedig az alkalmazás az egyik féle kliens szerepét tölti majd be. Egy ilyen elgondolásnak a megvalósításához számos MQTT kommunikációt megvalósító osztály könyvtár lehet segítségére az embernek fejlesztés során, és az ezalatt használni kívánt könyvtár meghatározását még korainak éreztem, mert ezen a területen a gyakorlatban történő megvalósítás során fog eldőlni, hogy melyik csomag a legmegfelelőbb az íródó program számára. Ettől függetlenül viszont a tervezett alkalmazás kommunikációs működésének alapelveit elméletben át lehet gondolni és elő lehet állni egy tervvel ezen a területen is.

Az elméleti síkon problémába nem futó alapelképzelések szerint az alkalmazás kommunikációs szempontból úgy fog működni, hogy amikor a felhasználó belép az alkalmazásba, az felveszi a kapcsolatot a szerverrel, és feliratkozik a tárolt védőmodulok témáira. Ezt a tárolt modulok azonosítóinak felhasználásával tudja megtenni, mert ahogy az már említésre került az irodalomkutatási fejezetben, az elképzelés szerint a figyelt tulajdonságok témái a „modulazonosító/tulajdonság” minta szerint fognak felépülni. Az alkalmazás használata során feliratkozó és publikáló szerepét is fogja érvényesíteni, előbbit a modulok tulajdonságainak témáira való feliratkozásával és az ezekre beérkező üzenetek feldolgozásával, utóbbit pedig a motor vezérlésének irányításával, aminek keretein belül a program egy a célzott védőmodul által értelmezhető üzenetet tudna küldeni a vezérlésre kijelölt témára. Abban az esetben, ha a felhasználó befejezi, vagy ha háttérbe kerül az alkalmazás, az kilépés előtt leiratkozik a figyelt témákról és bontja a kapcsolatot a szerverrel. Ami pedig a témák háttérbeli megfigyelésének területét illeti, a terv alapján az alkalmazás időnként fel fog ébredni egy időre, ami során újra felveszi a kapcsolatot a szerverrel, valamint a háttérben figyelni kívánt témákra feliratkozik, aztán

az időkerete lejárta után leiratkozik a témákról, bontja a kapcsolatot a szerverrel, aztán visszakerül alvó állapotba.

Véleményem szerint egy ilyen működési elvekre épülő kommunikációs rendszer jól ki tudja szolgálni a korábban megfogalmazott feladatteljesítési igényeket.

5 Tervezési munkafázisok, tapasztalatok és a rendszerterv

5.1 Tervezési fázisok és tapasztalatok

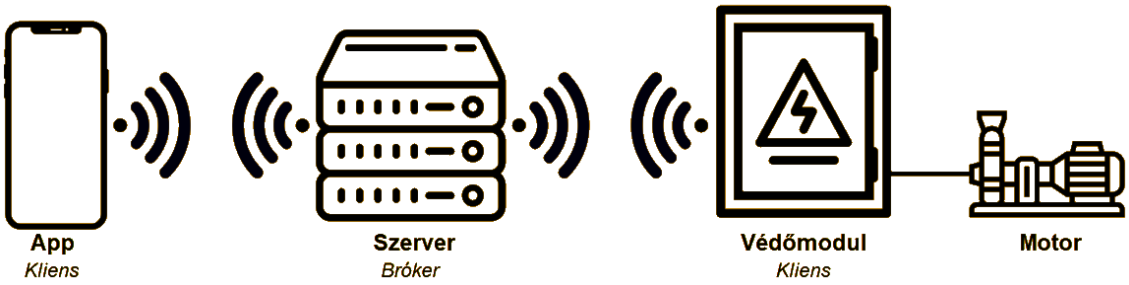
Ahogy az említésre került az igényspecifikációt leíró fejezet elején, számomra az alkalmazás, vagy rendszer megtervezése mondhatni már az igények megfogalmazásánál kezdődik, hiszen akkor körvonalazódnak a kívánt szoftver alapvető feladatai. Az ilyen feladatok köré, valamint ezeket észben tartva érdemes a rendszer konkrét működésének tervét felvázolni a rendszerterv készítése során. Abban az esetben, ha ezt az elvet követik a tervezők, akkor mire valóban elkészül a rendszer, az a lehető legegyszerűbben tudja majd elvégezni a fő feladatait, miközben használata egyértelmű marad a felhasználó számára.

Az alapfeladatok és felhasználói igények leírása alapján már egy kezdetleges látványterv és funkciólista is felállítható, azonban a pontosított, technikai szintű tervezés, majd később a tervek megvalósítása során ezek természetesen tovább bonyolódnak. A bonyolódások esetleg logikai hibákként a feladatok végrehajtásában, vagy technikai korlátokként a megvalósításhoz használt eszközökben és technológiákban jelenhetnek meg. Ilyen problémák fennálltakor először érdemes azt megállapítani, hogy a készülő rendszer feladatai mennyire függnek a probléma forrását jelentő feladattól, vagy folyamattól. Ennek megállapítása segíthet mérlegelni a tervezőknek, hogy milyen módon álljanak neki az adott nehézség megoldásának. Az ideális eset az, amikor a problémás folyamattól nem függ a többi, így azt akár alapjaitól indulva is teljesen újragondolhatja a fejlesztő. A nehezebbik féle az, amikor a megoldandó problémát hordozó feladat függelékeinek száma a teljes rendszer feladatainak számához képest aránylag magas. Ilyen nehézségbe a tervezés, vagy majd a megvalósítás elején jó belefutni, hogy a fejlesztő minimalizálni tudja az újragondolandó feladatok mennyiségét. Én a fejlesztési fázisban találkoztam ilyen nagy függelékű nehézségekkel, azonban annak ellenére, hogy a problémás feladatot sok más folyamat használta a rendszerben, mégis azokat sikerült úgy elszeparálni, hogy egy fejlesztési minta alapján történő újradolgozással sikerült megoldani a hibát. Erről azonban bővebben a megvalósításról szóló fejezetben lehet olvasni.

Tehát összegezve a szoftver tervezésének lépései számomra a feladatok specifikációjával kezdődtek, amik az eredeti igénylefektetést és problémafelvetést követően lettek megállapítva. Ezen elvárások tiszta közlését követte a kezdetleges menürendszer tervezetének felvázolása, ami lényegében az elvárt feladatok elérhetővé tétele volt a felhasználó számára úgy, hogy a felület egyértelműsége is szem előtt volt tartva. Ezek után a feladatok elvégzésének elméletbeli módszereinek tervezésébe fogtam, amik később, a megvalósítás során kerültek nagyobb fokú pontosítás alá.

5.2 Rendszerterv

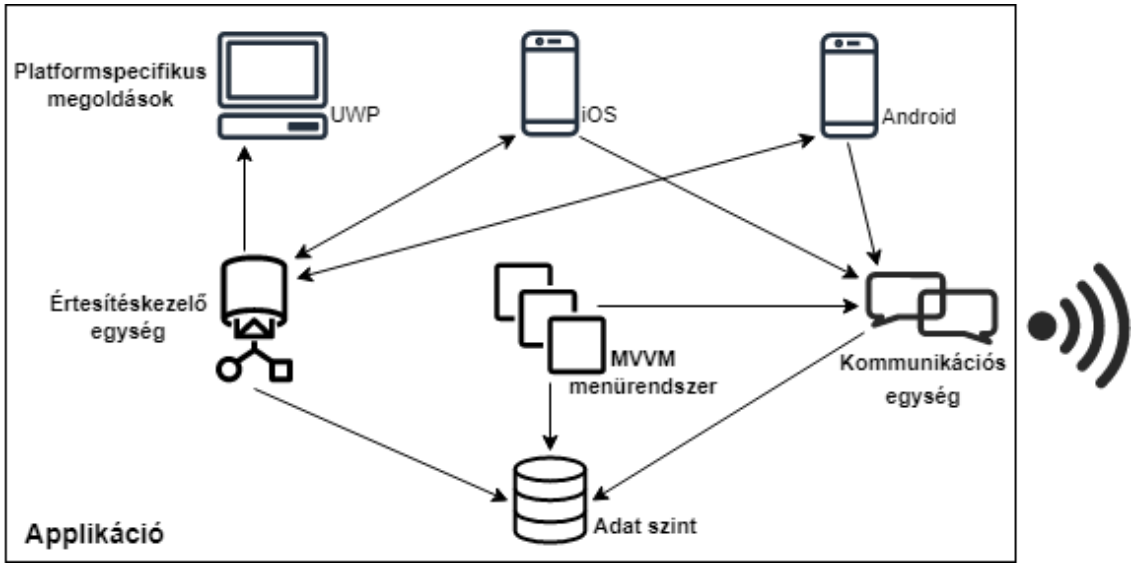
A rendszerterv célja az igényspecifikációban olvasható igények alapján megfogalmazott feladatokat végrehajtó komponensek struktúrájának leírása, valamint az alkalmazás teljes rendszeréről egy egységes tervet alkotni.



5.1. ábra. A teljes kommunikációs hálózat

Ahogy az az 5.1. ábrán is látható, az általam tervezett platformfüggetlen alkalmazás célja a teljes, azaz a védőmodulokból, az MQTT kommunikációs kapcsolatért felelős szerverből és az alkalmazásból felépülő rendszerbe való beépülés, mint MQTT kliens.

5.2.1 Architektúra



5.2. ábra. Az applikáció architektúrája egységekre bontva

Az alkalmazás architektúrájának szintjeit tekintve alul az adatbázis, és az ezt kezelő, azaz olvasó és író szerv helyezkedik el.

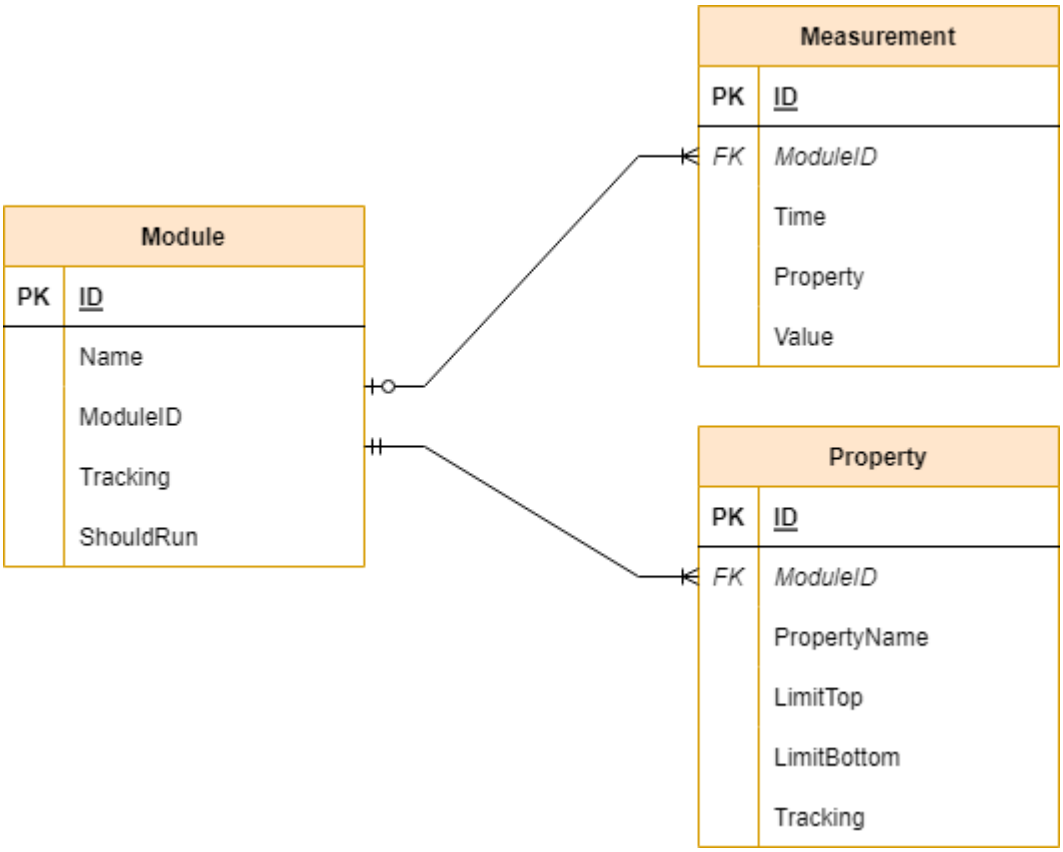
Ezt három struktúra használja. Első az MQTT klienst kezelő kommunikációs egység, ami felel az alkalmazás kliensi szerepének ellátásáért. A második az MVVM, vagy teljes nevén Model-view-viewmodel architekturális mintára épülő osztályszerkezet, ami az

alkalmazás felhasználói felületét, valamint ennek logikáját foglalja magába. A harmadik pedig a platformfüggetlen értesítéskezelő egység.

Végül pedig a szoftver felépítésének a tetején a platformspecifikus egységek foglalnak helyet, amik külön-külön iOS, Android és UWP specifikus megoldásokat tartalmaznak.

5.2.2 Adat szint

Az alkalmazás lokális adatbázisa három adattáblával rendelkezik, ezek a modulokat, a beérkezett mérési adatokat, valamint a modulok tulajdonságait tárolják.



5.3. ábra. Az adatbázis E-K diagramja

Ahogy ez az 5.3. ábrán található E-K, azaz egyed-kapcsolat diagramon is szerepel, egy védőmodult képviselő tipikus rekord rendelkezik egy azonosítóval, névvel, amit majd a felhasználó adhat meg szabadon, és magával a modul azonosítójával, amit a kommunikációs csatorna felépítéséhez fog használni. Továbbá tartalmazza a Tracking elnevezésű logikai változót, aminek az értéke alapján dönti el az alkalmazás, hogy küldhet-e értesítést az adott modul nevében, valamint a ShouldRun nevű, második logikai változót, aminek az értékéből azt az információt tudja levonni a szoftver, hogy a motornak működésben kellene-e lennie, vagy sem.

A modul elem azonban magában nem elég az összes információ tárolására, amit egy ilyen eszközről ismernie kell a rendszernek, ezért lett a Property, azaz tulajdonság elemeket

tároló tábla létrehozva. Ahogy az 5.3. ábrán látható, a modul és tulajdonság kapcsolata egy a többhöz alapon működik, ahol a program helyes működésének érdekében mindkettő rekordtípus számára létfontosságú, hogy a tulajdonság elemben legyen egy modulra mutató adattag. A tulajdonság elem egy adatbázisban szereplő modul egyik megfigyelt tulajdonságát reprezentálja, ezért rendelkezik egy saját azonosítóval, a már korábban említett tulajdonos modul azonosítójával, ModuleID néven, és az eszközhöz tartozó egyik tulajdonság nevével, PropertyName néven. Az utolsó három adattagból pedig az első kettő az adott tulajdonság felső, valamint alsó határértékeit jelenti, amik által képzett tartományon, ha egy beérkezett adat kívülre esik, akkor az alkalmazásnak értesítést kell küldeni, amennyiben a körülmények megfelelőek.

A Tracking, azaz követés elnevezésű utolsó adattag pedig egy logikai változó, ami alapján a kommunikációs egység eldönti, hogy feliratkozzon-e a tulajdonság témájára az MQTT hálózatban. A távoli elérésű védőmodul működése alapján, az adatbázisban tárolt modul elemekre egyenként négy darab tulajdonság rekordnak kell jutnia. Ezek alatt értendő a be- és kimenő feszültségek, a bemenő áramerősség és az aktuális adat arról, hogy jár-e a motor, vagy nem. Ezeknek a tulajdonságoknak, említés szerinti sorrendben a VIn, VOut, CIn és Running nevek lettek adva, és az adatbázisban is így szerepelnek.

A harmadik táblában pedig a Measurement, azaz mérés elnevezésű rekordokat tárolja a rendszer. A mérési elemek a beérkezett MQTT üzenetekből kinyert adatokból vannak összeállítva, és tartalmazzák a saját azonosítójukat, annak a modulnak az azonosítóját, amihez tartoznak, a mérés elmentésének időpontját, a mért tulajdonság nevét, valamint magát a mért adatot. A modul és mérés között is egy a többhöz típusú kapcsolat van, azonban annyiban különbözik a modul és tulajdonság közötti kapcsolattól, hogy a jelenleginél csak a mérési egyednek elengedhetetlen, hogy legyen a hozzá tartozó modulra mutató adattag. A modul oldalán ez a kapcsolat opcionális, mivel a mérések létezéséért nem az alkalmazás felel, az csupán fogadja az üzeneteket, amelyekből értelmezésük után létrehozza a mérési elemeket.

Az adatbázist létrehozó, író és olvasó adat szint megvalósításához az *sqlite-net-pcl* nevű osztály könyvtárat használom.

5.2.3 Kommunikációs egység

Ahogy az már ismertetve volt, az alkalmazás egy kliens szerepét tölti be a védőmodulokat kezelő MQTT hálózatban. Feladatai kliensi szempontból a következők. A hálózatra való fel- és lecsatlakozás. A figyelni kívánt védőmodulok publikációs témáira való fel- és leiratkozás. A beérkező üzenetek értelmezése. Üzenetek publikálása.

Egy tipikus téma, amire a program feliratkozik a „modulazonosító/tulajdonság” mintát követi. Ebből kifolyólag a fel- és leiratkozási feladatok sikeres elvégzéséhez a kommunikációs egységnek szüksége van az adat szint által szolgáltatott, tárolt tulajdonságokra. A tulajdonság rekord ModuleID és PropertyName elnevezésű

adattagjaiból fel tudja építeni a témát, amire fel kell iratkoznia, de a feliratkozást csak akkor végzi el, ha a rekord követést jelző adattagja igaz állapotban van. Ez azonban nem fedi le az összes témát, amire fel kell iratkoznia, mivel a program feladatai közé tartozik a szerver által naplózott adatok fogadása és feldolgozása. Ilyen esetben a feliratkozási téma a következő mintát követi: „modulazonosító/logback”. Ilyen esetekben a „logback” szó állandó, azaz nem reprezentatív jelleggel szerepel a mintában, ahogy a modulazonosító.

A beérkező üzenetek értelmezése az előző, fel- és leiratkozási feladat leírásában olvasható módszerhez hasonló, azonban ebben az esetben nem a téma összeépítése a feladat része, hanem egy már meglévő témában található adatok felhasználása. Egy MQTT üzenet rendelkezik témával, valamint payload-dal, azaz rakománnyal. Ez a rakomány a modul által mért, majd elküldött érték, amit a kommunikációs egységnek kategorizálni kell, ezért a témát az előzőekben ismertetett minták szerint felbontja, hogy kinyerje belőle a küldő modul azonosítóját és a tulajdonság nevét. Az így feldolgozott téma alapján a rakomány kategorizálhatóvá válik és az értelmezés ezzel elvégezhető. Az értelmezés megtörténte után pedig az alkalmazás tudja használni az így előállt adatot más feladataira.

Az kliensi feladatokat ellátó kommunikációs egység megvalósításához az *MQTTnet* nevű osztály könyvtárat használom.

5.2.4 Menürendszer

Az alkalmazás felhasználói felületéért és fő logikájáért felel a menürendszert képező osztályszerkezet, aminek a tervezéséhez az MVVM, azaz magyarul modell-nézet-nézetmodell nevű architekturális mintát használtam.

A mintában meghatározott nézet alatt a felhasználói felület értendő, amely menürendszerének főbb elemeiről készült látványterve az igényspecifikációról szóló fejezetben, a 4.2. ábrán látható. Amíg a nézet réteg célja az adatok megjelenítése, addig a nézetmodell réteg feladata a megjelenítendő adatok előkészítése, kezelése, esetekben ezen adatok felülírása. Ehhez az osztályszerkezet ezen rétege használja az alkalmazás adat szintjét. Végül, de nem utolsósorban pedig itt található a 4.1. ábrán látható, használati eset diagramon leírt feladatok elvégzésének logikája. A nézet számára a nézetmodell által szolgáltatott adatokat pedig a modell szabja meg. A program adat szintjén található adatbázis is a modell által meghatározott modul, tulajdonság és mérés mintájú rekordokat tárol.

5.2.5 Értesítéskezelő egység

A tervezés korai fázisai során már megállapítottam, hogy a lokális értesítéseknek platformspecifikus megoldásokra van szükségük. Az alkalmazás logikájának legnagyobb része azonban így is a platformfüggetlen, osztott kód kategóriába tartozik, ezért a programnak szüksége van egy az értesítéséket kezelő egységre, ami összeköti a

platformspecifikus részeket az alkalmazás többi részével. Ezen túl pedig ez a harmadik egység a rendszer tervében, ami az adat szintet használja a feladatának elvégzésének érdekében.

Fő feladata a már értelmezett beérkezett üzenetek vizsgálata, hogy eldöntse, hogy szükséges-e értesítést küldeni a felhasználónak.

5.2.6 Platformspecifikus megoldások

Ahogy az előző alfejezetben is említésre került, a tervezés korai fázisai során már látszott, hogy szükség lesz platformspecifikus megoldásokra. Az ilyesféle egységek által megoldandó feladatok közé tartozik a háttérmunkálatok elvégzése, valamint a lokális értesítés összeállítása, és kiküldése.

Mivel az alkalmazás UWP verzióját asztali számítógéppel fejben terveztem használni, ezért ennek az egységnek egyedül a lokális értesítések létrehozása és kiküldése a feladata. A háttérmunkálatok kihagyása mellett azért döntöttem, mert az asztali számítógépen, ha más tevékenységet kívánna végezni a felhasználó, akkor egyszerűen leküldheti tálcára az alkalmazást, és az ugyanúgy fog működni továbbra is, csupán nincs a felhasználó előtt.

Android és iOS platformokon viszont már az értesítések összeállítása és kiküldése mellett szükséges a háttérmunkálatok elvégzése is, ugyanis a feladatok megfogalmazása között szerepelt, hogy az alkalmazás háttérben lévő állapotban is képes kell, hogy legyen beérkező adatot menteni, valamint vészhelyzet esetén értesítést küldeni. Ehhez ideális esetben az szükséges, hogy időnként felkeljen az alkalmazás, figyelje egy ideig a beérkező adatokat, azokat értelmezze, majd vizsgálja meg azokat, hogy szükséges-e egy értesítés kiküldése.

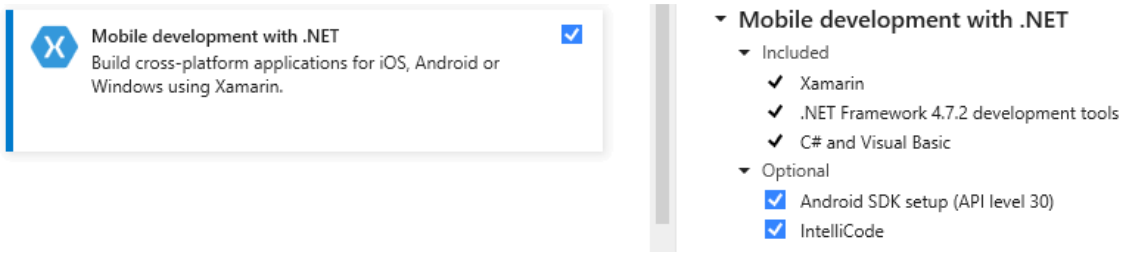
Ezekhez a platformspecifikus megoldásokhoz a megvalósításuk során csak az UWP értesítések kiküldéséhez lesz szükség osztály könyvtár használatára, ez pedig a *CommunityToolkit.WinUI.Notifications* nevet viseli.

6 Megvalósítás

Az alkalmazás rendszertervében részletesen ismertetésre kerültek az architekturális egységek főbb feladatai, és a feladatok átgondolása jó alappal szolgált a megvalósításhoz is, ahol a lefejlesztendő feladatok többsége előre látható volt.

6.1 Fejlesztési környezet

Az irodalomkutatást bemutató fejezet során a kiválasztott megoldási módszereket leíró alfejezetben a Xamarin keretrendszer használatára esett a választásom. Ahogy az a 6.1. ábrán is látható, a keretrendszernek és eszközeinek telepítése egyszerűen elérhető a Visual Studio installáló alkalmazásának felületén. Az alkalmazás megvalósításához azért ezt a fejlesztői környezetet választottam, mert az egyetemi tanulmányaim során ebben szereztem a legtöbb tapasztalatot.



6.1. ábra. Xamarin a Visual Studio telepítő alkalmazásában

6.2 Adat szint

Az adat szint elkészítéséhez a már korábban említett *sqlite-net-pcl* osztály könyvtárat használtam. A könyvtárat több internetes tanulóanyag forrás között a Microsoft hivatalos blogja is ezt használja. [21]

```
public const string DBFilename = "ModuleClientData.db3";

public const SQLite.SQLiteOpenFlags Flags =
    SQLite.SQLiteOpenFlags.ReadWrite |      // read/write mode
    SQLite.SQLiteOpenFlags.Create;          // create the db if it doesn't exist
```

6.2. ábra. Az adatbázis konfigurációja

Ahogy az a 6.2. ábrán is látható, az adatbázis létrehozásához és konfigurálásához szükséges számos paraméter megadása, ezek közé tartozik az adatbázis neve és néhány kezelési módot jelző enumerátor megválasztása. Én az alkalmazás lokális adatbázisához ezt a két paramétert állítottam be, hogy az írható és olvasható legyen, valamint, hogy a program első futtatásakor, vagy ha a felhasználó úgy döntene, hogy új példányt szeretne, és kitörli a fájlt, akkor hozzon létre egy új adatbázist

Ebben az osztályban a feladatok nagyobbik részét a sztenderd CRUD műveletek metódusainak hívása tette ki, azonban a különböző táblák elkülönítése funkcionális szempontból hasznos, hogy később a modulokat módosító osztályok egyszerűbb

metódusai ne érjék el fölöslegesen például a mérések tábláján módosításokat végző metódusokat. Ennek érdekében létrehoztam az adatbázis CRUD, azaz létrehozó, olvasó, frissítő és törlő műveleteinek függvényei alapján elkülönítő interfészeket, amiket az adatbázis kezelő osztály megvalósít. Ezek után pedig implementáltam egy egyszerű factory, azaz gyár osztályt, ami az előbb említett interfészeken keresztül adja vissza az adatbázis kezelő osztály objektumát, hogy az így átadott példány valóban csak azok a feladatok elvégzésére legyen képes, amelyre az adott helyzetben szükség van.

Az alapvető feladatok egyszerűségével ellentétben a beérkezett, már részben a kommunikációs egység által kategorizált mért adatok további értelmezése is erre az osztályra jut feladatként, mivel a beérkezett adatot el is kell menteni. A kommunikációs egység a beérkezett üzenetből egy olyan objektumot épít fel, ami tárolja a küldő modul azonosítóját, a tulajdonság nevét, valamint magát a mért értéket. Azonban az objektum tulajdonság értéke alapján két féle módon kell tudnia kezelnie az alkalmazásnak a beérkező mért értéket.

Egyszerűbb körülmények között a téma szimplán a tulajdonság, és ilyenkor rögtön menthető is a beérkezett érték az adatbázisba. Abban az esetben viszont, ha az üzenet témája a „logback” szó, akkor a szervertől lekért, naplózott adatot tartalmazó üzenetről van szó. Emiatt az üzenet rakományát is bontani kell, hogy a program kinyerhesse belőle az elmentendő értékeket. Alapesetben az üzenet rakománya átmegy egy formátum ellenőrzésen, és már menthető is, de az ilyenkor fogadott érték ezt a mintát követi: „azonosító,dátum,VInérték,VOutérték,CInérték,Runningérték”. Ebben a mintában szerepelnek a szervertől lekért mérések, és naplózásuk időpontja, hogy a felhasználó is tudja, mikor mérte a modul az adatokat. Ilyen rakományok formájában juttatja el a szerver a naplózott adatokat az alkalmazás felé mentési időpontok szerint felosztva egy-egy üzenetként. Az értékek és naplózásuk időpontjainak kinyerése és ellenőrzése után a metódus átadja az adatokat az adatbázisnak mentésre.

6.3 Kommunikációs egység

Egy MQTT szerverrel való kapcsolatfelvételhez szükségesek bizonyos paraméterek, mint például a bróker címe és kommunikációs portja. Ezeket az értékeket, mint az adatbázis által használt paramétereket is egy globális változókat tartalmazó segédosztály szolgáltatja.

A kommunikációt megvalósító osztály már használja az adat szintet, azon belül is a tulajdonságokat tároló adattáblát, ezért az inicializálás során kér a gyár osztálytól példányt a tulajdonságok tábláját kezelő interfészre.

Minden alkalommal, amikor a program előtérbe kerül, legyen az bezárt vagy alvó állapotból való indítás a kommunikációs egység kikéri a tárolt tulajdonságokat az adatbázistól, aztán végig iterál rajtuk, és mindegyiknél egyenként, adataik alapján összeépít egy témát a már emlegetett normál minták, valamint utána modulonként a

naplózási minta szerint. Az adott felépített mintákra pedig feliratkozik, ha az adott vizsgált tulajdonság Tracking adattagja igaz állapotban van. Az automatikus feliratkozási funkció ezen felül a modul és tulajdonság rekordokat tároló adattáblák változásai után mindig lefut, hogy új modul felvételének esetén rögtön figyelni tudja a modul témáit. Ezzel ellentétben, minden háttérbe, vagy kikapcsolt állapotba kerülésekor ugyanezzel az összeépítési módszerrel átmegy a tulajdonságokon, azonban ilyenkor leiratkozik a témákról.

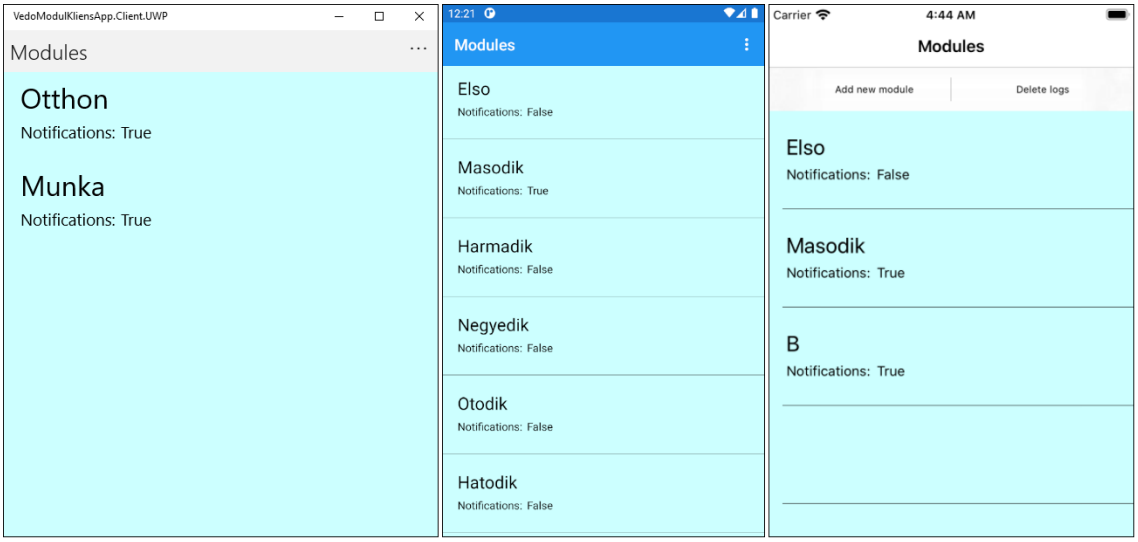
Végül pedig ez az egység végzi el a beérkező üzenetek értelmezésének első felét, azaz a mérés moduljának, valamint tulajdonságának szétbontását.

6.4 Menürendszer

A menürendszer osztályszerkezete a már korábban emlegetett MVVM minta alapján készült, és ennek megvalósításához a Can Bilgin által írt „Mobile Development with .NET” című könyvet használtam segítségként. A könyv számomra kellően részletesen bemutatta egy jó nézetmodell tipikus felépítését és feladatait, valamint a felhasználói felületek különféle tervezési lehetőségeit a Xamarin keretrendszerben. [22]

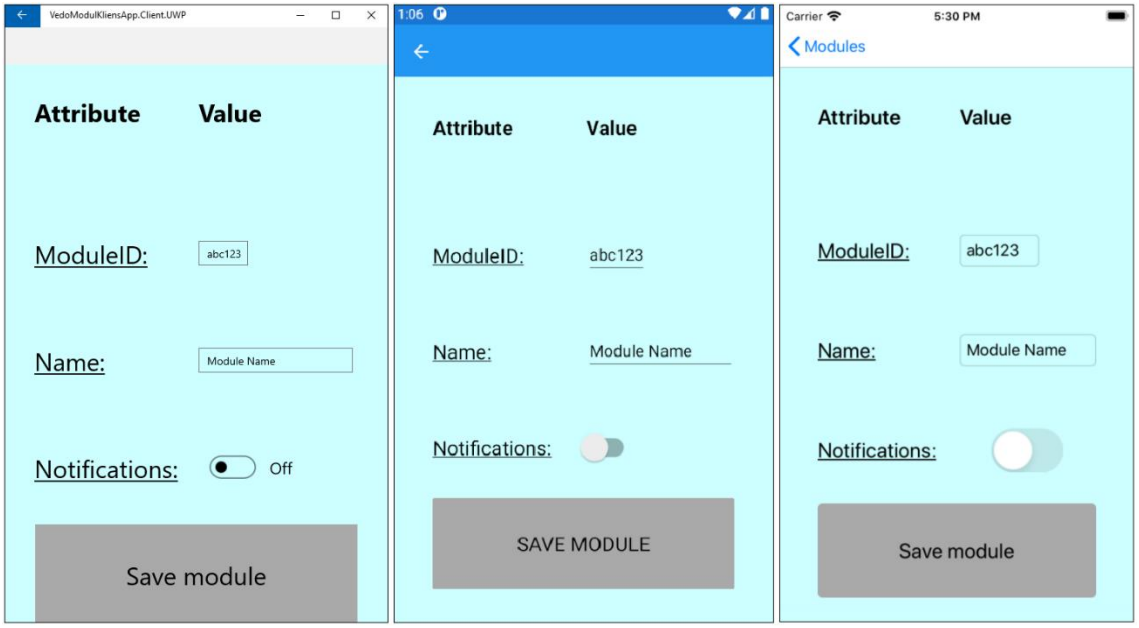
6.4.1 Felépítés és navigálás

Eredetileg az igényspecifikáció során fogalmaztam meg azt az elvárást az alkalmazással szemben, hogy a menürendszer navigálása szűkítse a fókuszt lépésről lépésre. Ennek az elvnek az észben tartásával fogtam hozzá a részletesebb felhasználói felület megvalósításához. Az elkészült szűkülő menürendszerben a nézetek a következőképp navigálhatóak egymás között: A 6.3. ábrán látható kezdő nézet a tárolt modulok listájával fogadja a felhasználót.



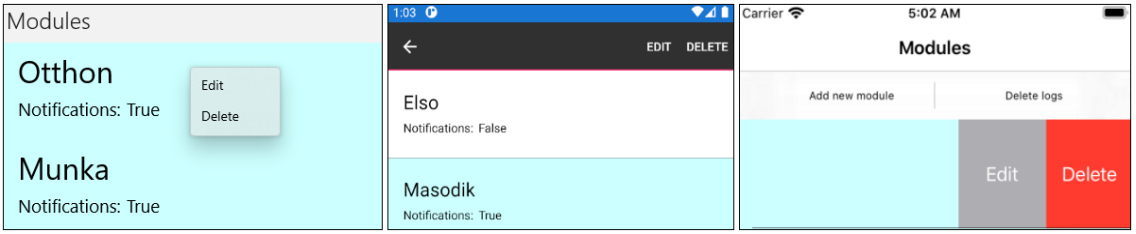
6.3. ábra. A listafelület UWP, Android és iOS platformokon

Itt az eszköztárból elérhető az új modul felvételének nézete, amit kiválasztva a felhasználó a 6.4. ábrán látható nézetre kerül, ahol megadhatja, az új modul nevét, modul azonosítóját, valamint, hogy kívánja-e, hogy az új modul küldhessen-e rögtön értesítéseket.



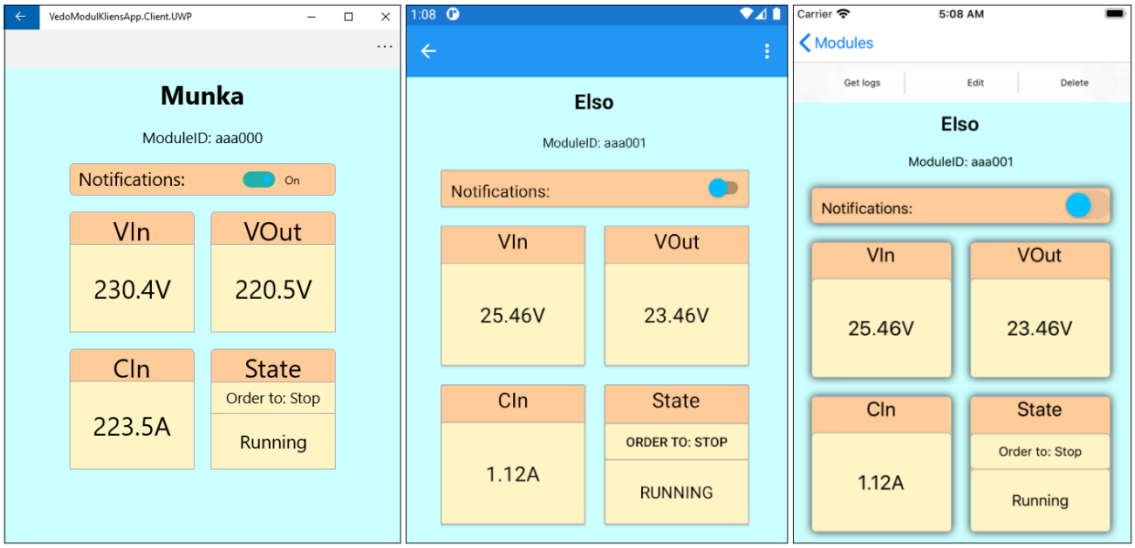
6.4. ábra. Modulfelvétel UWP, Android és iOS platformokon

Mentés, vagy visszalépés esetén a felhasználó ismét a modulok listanézetén találja magát, ahol egy tetszőleges modul 6.5. ábrán található kontextus menüjét megnyitva a felhasználó találkozik két újabb lehetőséggel, amik a kiválasztott modul törlése, vagy a modul szerkesztése.



6.5. ábra. Kontextus menü UWP, Android és iOS platformokon

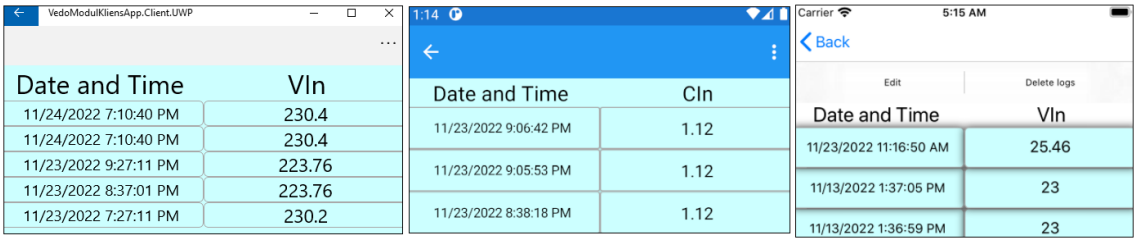
A kijelölt modul szerkesztésének esetén a korábbi, új modul felvételére használt nézeten találja magát a felhasználó, azonban most a kitölthető részekben már szerepelnek az adott modul adatai, ilyenkor a mentés az adott modult frissítené. Mentés, vagy visszalépés esetén ismét a listanézetre jut vissza a használó, ahol ha kiválaszt egy modult a listából, akkor a 6.6. ábrán látható, adott modul megfigyelt tulajdonságainak nézetére kerül.



6.6. ábra. Tulajdonságok nézete UWP, Android és iOS platformokon

Ezen a nézeten láthatóak a tulajdonságok legutóbb beérkezett mért értékei, beleértve a motor működési állapotát, és a motor irányítására szánt gombot, amit lenyomva annak a hozzátársított *Command*-ja a kommunikációs egység használatával küld egy egyszerű igaz vagy hamis jelet a „modulazonosító/Order” témára. A jel értéke a motor működésének mindenkori legutóbbi állapotának ellentéte, tehát ha a motor működésben van, ezzel a gombbal arra lehet utasítani, hogy kapcsoljon ki, abban az esetben pedig, ha le van állítva, akkor küldjön rá indító jelet. Itt is találhatóak az eszköztárban gombok, ezekből kettő ugyanazzal a törlő vagy szerkesztő funkcionalitással bír, mint az egyes modulok kontextusmenüjében található gombok a modulok listájának nézetén. A harmadik pedig törli a mentett méréseket, majd lekéri az adott modul szerveren naplózott adatait.

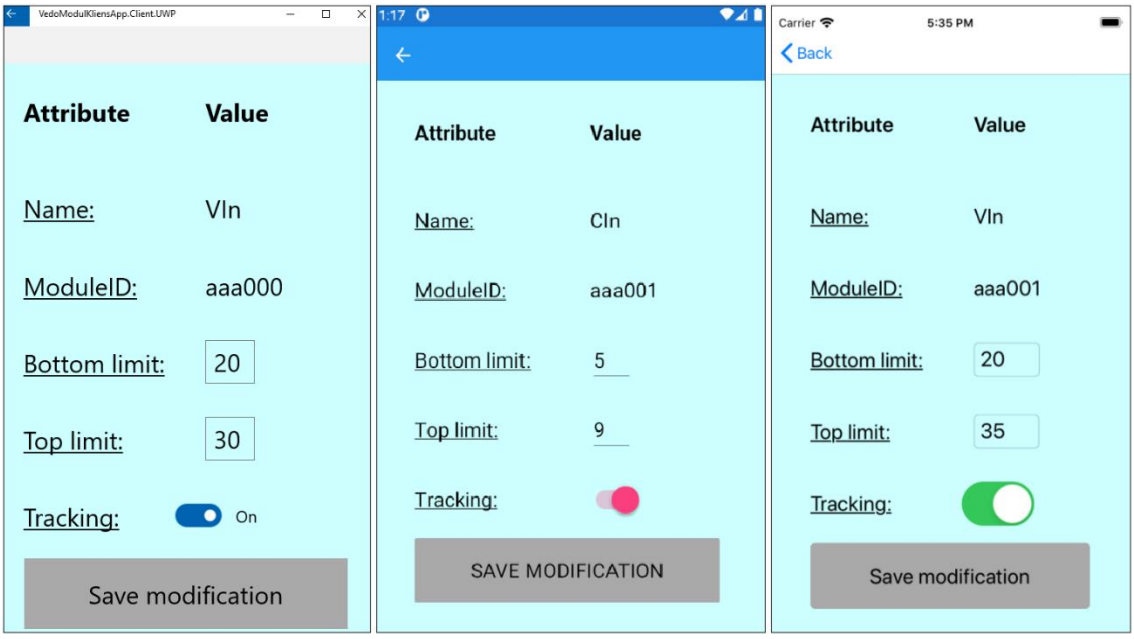
Ha egy tulajdonság gombjára kattint a felhasználó, akkor eljuthat a 6.7. ábrán található adott tulajdonság méréseinek listájára, ahol láthatja a lokális adatbázis által tárolt méréseket, és a pontos dátumukat és időpontjukat.



6.7. ábra. Tulajdonság méréseinek listája UWP, Android és iOS platformokon

Itt az eszköztár gombjai között találhatóak az adott tulajdonság szerkesztéséért felelős nézet megnyitásának és a naplózott adatok törlésének gombjai. A szerkesztő gombra kattintva a felhasználó a 6.8. ábrán található nézetre kerül, ahol látni a szerkesztett tulajdonság moduljának nevét, azonosítóját, valamint az adott tulajdonság alsó és felső

határértékeit, amik szerkeszthetőek. Ezen felül itt állítható be, hogy a tulajdonságra fel akarja iratkoztatni-e a használó az alkalmazás kommunikációs egységét vagy sem. Ez a legmélyebb, vagy legszűkebb fókuszú menü, ameddig a felhasználó el tud jutni.



6.8. ábra. Tulajdonság szerkesztése UWP, Android és iOS platformokon

6.4.2 Működése

A nézetek megvalósításánál a fő cél az, hogy a konkrét „xaml” kiterjesztésű, deklaratív jellegű, felhasználó által látott oldalak code-behind, azaz mögöttes kód logikájában ne történjen adatmanipuláció, vagy végrehajtás. A feladatok eszerint történő elválasztása tiszta, karbantartható kódot eredményez, ami megfelel az MVVM mintában előírtaknak. Ebből kifolyólag pedig a mögöttes kódban található metódusok, amik többnyire az adott oldal eszköztárában helyet foglaló gombok lenyomásakor meghívódó események kezeléséért felelősek, navigációs kódot futtatnak, vagy a saját nézetmodelljükben található éppen szükséges metódusokat hívják meg.

Ezen túl annak érdekében, hogy az alkalmazás az adatok betöltése során ne akassza meg a fő, felhasználói felület betöltéséért felelős szálat, a mögöttes kódban felülírható *OnAppearing*, magyarul a betöltés során lefutó metódusban történik a megjelenítendő adatok lekérése. Ezek az adatlekérések adatbázis hívásokat jelentenek, ezért ha a fő szálon lennének meghívva, akkor a felhasználói felület betöltésének meg kellene állnia, amíg meg nem érkezik az adat az adatbázistól, ezzel animációs képkockákat veszítene a kirajzolás, így akadozóvá téve az alkalmazást. A betöltés során lefutó metódusokban történő adatfrissítés hívása azonban elméletben nem kellene, hogy elég legyen az akadozás megállításához, viszont azért működik, mert a frissítésért felelős *Command* típusú tulajdonságok , amiket ezek a metódusok meghívnak úgy futtatják a saját frissítő metódusukat, hogy azt, mint *Task-ot*, azaz feladatot átadják a *ThreadPool-nak*, azaz a

szálkészletnek, hogy a háttérben, a fő száltól függetlenül végezhessék el az adatok lekérését, és amikor ez megtörtént, frissítsék a már betöltött felületen látható, nézetmodellhez dinamikusan kapcsolt tulajdonságokat. Ilyen, akadozásmentes frissítéshez használt metódusok hívás sorendjének egy példája a 6.9. ábrán látható, amin az adatok frissítésének végrehajtása narancssárga színnel van aláhúzva.

```
protected override void OnAppearing()
{
    (BindingContext as ModuleListViewModel).OnPageAppearing();
    base.OnAppearing();
}
```

```
public void OnPageAppearing()
{
    EventSingleton.Instance.ModuleDbChanged += RefreshModulesAndConnection;
    RefreshCommand.Execute(null);
}
```

```
RefreshCommand = new Command(() =>
{
    Task.Run(RefreshModulesAndConnection);
});
```

```
private async Task RefreshModulesAndConnection()
{
    if (isRefreshing)
    {
        return;
    }
    IsRefreshing = true;
    modules = null;
    List<Module> moduleList = await moduleDB.GetModulesAsync();
    Modules = new ObservableCollection<Module>(moduleList);
    IsRefreshing = false;
    await commService.Connect();
}
```

6.9. ábra. Oldal betöltésekor induló frissítés metódus hívási lánca

6.5 Értesítéskezelő egység

Az értesítésekért felelős, még a platformfüggetlen kategóriába tartozó osztálynak három feladata van.

Az első a három platformfüggetlen értesítéskezelő osztályokkal való kapcsolat felállítása. Ezt egy értesítésküldő metódust definiáló interfész létrehozásával és a különböző platformokon történő implementálásával, utána pedig az osztályok *DependencyService-hez* való hozzáadásával lehet megoldani. Így szükség esetén az interfészt megadva lehet kérni az adott futó projekt platformjának megfelelő példányt a *DependencyService* használatával.

A második feladat az értesítésekkel ellátott modulokból, ezeknek az azonosítóiból, valamint a figyelni kívánt tulajdonságokból álló három lista vezetése és frissítése a modul és a tulajdonság adattáblák változásakor. Ez a frissítő metódus látható a 6.10. ábrán.

```
private static async Task RefreshTrackingLists()
{
    List<Module> moduleList = await moduleDB.GetModulesAsync();
    trackedProperties = await propDB.GetPropertiesAsync();

    // get the moduleIDs of the tracked modules
    trackedModules = moduleList.Where(m => m.Tracking).ToList();
    trackedModuleIDs = trackedModules.Select(m => m.ModuleID).ToList();
    // get the properties that are tracked and belong to tracked modules
    trackedProperties = trackedProperties.Where(p => trackedModuleIDs
                                        .Contains(p.ModuleID) && p.Tracking).ToList();

    System.Diagnostics.Debug.WriteLine("notif: Tracked properties list refreshed!");
}
```

6.10. ábra. A figyelt tulajdonságok listájának lekérdezése

Erre a listára a harmadik feladatának elvégzéséhez van szüksége, ami a beérkező üzenetek kiértékelése, és szükség esetén értesítés küldése. Először is az értesítésekkel ellátott modulok listáját fésüli át, és ha a beérkezett üzenetet küldő modul az értesítéses listán megtalálható, akkor kvalifikált arra, hogy a beérkezett értéket is átvizsgálja. Ebben az esetben kipróbálja, hogy az érték megfelelő formátumú-e, aztán azt nézi meg, hogy az adat a tulajdonságának a felhasználó által megszabott határértékei által behatárolt tartományon belülre esik-e. Abban az esetben, ha ezen a tartományon kívülre esett, akkor a tárolt platformspecifikus értesítéskezelő segítségével értesítést indít, megadva a problémára futó tulajdonságot és a modul azonosítóját, majd végül módosítja és frissíti a modult, hogy mostantól ne kapjon értesítéseket, a Tracking tulajdonság visszaállításig.

6.6 Egységek kommunikációja

Az utóbbi alfejezetekben ismertetett egységek mellé szükséges egy olyan szerv, ami értesíti azokat, hogy szükségessé vált egy feladat elvégzése. A programnak egyfajta értesítésre van szüksége ahhoz, hogy például a felhasználói felületen az éppen megtekintett modul adott tulajdonsága frissüljön, amikor beérkezik egy ilyen adatot hordozó üzenet. Az egységeket értesítő funkciónak dinamikusnak kell lennie, hiszen az előző példával élve, a tulajdonságnak csak akkor kellene felülírnia az étékét, ha az valóban meg van jelenítve, különben feleslegesen hajtaná végre a frissítést. Annak érdekében, hogy ténylegesen csak az a nézet frissüljön, ami éppen meg van jelenítve, az egységeket értesítő, azaz a feladatok metódusait meghívó szervnek egy hívási listával kell rendelkeznie, amire fel és amiről le is lehet iratkoztatni feladatokat.

A felvázolt probléma megoldásához, tehát az éppen aktuális feladatok elvégzésére hívásához event-eket, azaz eseményeket használtam. Az helyes működéséhez az alkalmazásnak három eseményre van szüksége, ezeket a ModuleDbChanged, a PropertyDbChanged és a MessageReceived beszédes nevekkel hoztam létre. Ezek

magyarra fordítva, sorrendben a ModulAdatbázisMegváltozott, a TulajdonságAdatbázisMegváltozott, valamint az ÜzenetÉrkezett neveket viselnék.

Az első, modulok adatbázisára vonatkozó esemény akkor következik be, amikor a modul rekordokat tároló adattábla változik, azaz amikor egy modult ment, módosít, vagy töröl a rendszer. Erre az eseményre a menürendszerből megjelenésükkor fel, majd róla eltűnésük során leiratkoznak a menürendszer modul listáját, valamint a modul tulajdonságait bemutató, nézetmodellt frissítő metódusok. Ezeken túl pedig a kommunikációs egység MQTT feliratkozásait frissítő és az értesítéskezelő egység figyelni kívánt egyedeinek listáját frissítő metódusok iratkoznak még fel.

A második esemény hasonlít az elsőre abban, hogy ez is az adatbázis változásához köthető, de ez a tulajdonságokat tartalmazó tábla módosításainak teljesítései után indul el. A tulajdonságok változását jelző eseményre az előző bekezdésben emlegetett kommunikációs egység, és az értesítéskezelő egység frissítő metódusai iratkoznak fel.

Végül pedig a harmadik esemény a szimplán akkor következik be, amikor a kommunikációs egység MQTT üzenetet érkezését észleli, azonban ez annyiban különbözik az előzőektől, hogy ez az esemény paraméterként hordoz egy objektumot, amibe a kommunikációs egység által részlegesen értelmezett üzenet adatai kerülnek. A példány adattagjait a modul azonosítója, a tulajdonság és a rakomány teszi ki. Az eseményre feliratkoznak a modul tulajdonságait frissítő, az értesítéskezelő egység üzeneteket kiértékelő és az adatbáziskezelő osztály beérkezett méréseket mentő metódusai.

A többszörös feliratkozásból kiinduló problémák elkerülésének érdekében a feliratkozni készülő metódusokat a rendszer megpróbálja le, majd csak aztán feliratkoztatni. Amennyiben pedig egy menürendszerhez tartozó metódusról beszélünk, akkor az adott oldal előtérbe kerülésekor feliratkozik, majd eltűnésekor leiratkozik az adott metódus.

6.7 Platformspecifikus megoldások

Ahogy az az értesítéskezelő egységnél is bemutatásra került, a specifikus értesítéskezelő osztályok megvalósítanak egy interfészt, ami egy értesítésküldő metódust deklarál, cím és üzenet paraméterekkel, amik az értesítésben megjelenő szöveget képezik. Az Android és iOS értesítéskezelő osztályok megvalósításához a Microsoft hivatalos tanulóanyagát használtam fel. [23]

6.7.1 Android

Android platformon a háttérmunkálatokhoz egy periodikus *Job*, vagy magyarul munka beállítása a megoldás, amivel a rendszer beütemezi egy metódust időnkénti végrehajtásra. A metódus újraütemezését le lehet állítani, valamint magát az ütemező háttérmunkát is le lehet mondani az arra használatos kóddal. Továbbá megadható, hogy a rendszer milyen

gyakran ismételje meg a metódus lefuttatását, egy rugalmassági időkerettel együtt, ami döntési szabadságot ad az operációs rendszernek, hogy a megadott kereten belül önmaga eldönthesse, mikor végzi el a munkálatot. Jelen esetben én 15 percenkénti elvégzésre állítottam be az időzítőt, 5 perces rugalmassági idővel.

A munkálat metódusa először inicializálja a szükséges egységeket, névlegesen a kommunikációs egységet, ami csatlakozik a brókerhez és feliratkozik a figyelendő témákra, valamint az értesítéskezelő egységet, ami a beérkező üzeneteket értékeli ki. Ezután a metódus vár 30 másodpercet, hogy mindenképpen legyen elegendő beérkező adat, aztán bontja a kapcsolatot a brókerrel, majd kilép.

Az alkalmazás ezután visszakerül alvó állapotba, amiben további legfeljebb 15 percig marad, a következő háttérmunkálat elindulásáig.

6.7.2 iOS

A háttérmunkálat ezen a platformon is gyakorlatilag ugyanazt végzi el, ezalatt értve az egységek inicializálását, majd a várakozást, egy vagy két sor platformspecifikus kód különbségével, amit a háttérmunkálat lezárásához használ mindkét platform.

A különbség a háttérmunkálat ütemezésén látszik meg. Először is iOS-en a háttérmunkálat beütemezését egy *Background Fetch*, magyarul háttér beszerzés nevű operációs rendszer funkció végzi el. A nagy különbség azonban az, hogy amíg az Android platform olyan szinten szabad kezet ad, hogy megszabhatóvá teszi a fejlesztőnek, hogy milyen gyakran fusson le a munkálat, addig az iOS nem ilyen bőkezű. Ezen a platformon az operációs rendszer határozza meg, hogy mikor tartja kedvezőnek a háttérmunkálatok elvégzését. Ebből kifolyólag valószínűleg nem lesz olyan konzisztens a háttérben fogadott adatok naplózása, mint az Android verzió esetében, de ez a háttérmunkálat még az ilyen, ideálistól eltérő, operációs rendszer által szabott korlátozással együtt is kielégíti az eredeti tervezett funkcióval szemben állított elvárásokat.

7 Tesztelés

7.1 Felhasznált eszközök

Az alkalmazás UWP verziójának tesztelését a saját asztali számítógépemen végeztem, amin Windows 10-es operációs rendszer fut. Az Androidos verzió két eszközön került tesztelésre: egy emulátoron, ami az operációs rendszer 11.0-ás verzióját, valamint egy saját eszközön, ami az Android 8.0-ás verzióját futtatja. Az iOS verzió virtuális gép segítségével, egy iOS 13.7-es operációs rendszer verziót használó emulátorral lett tesztelve.

Annak érdekében, hogy az adott verziójú alkalmazásban futó folyamatok megfigyelhetőek legyenek, a debug, azaz a hibakereső kimenetre író kóddal láttam el az alkalmazás tesztelendő feladatait. Az ilyen jellegű rövid szöveg kiíratások alapján leolvashatóvá váltak az alkalmazás felhasználói felületét nem befolyásoló feladatai és folyamatai. Továbbá ez a tesztelési módszer meglehetősen hasznosnak bizonyult az alkalmazás megvalósítása során is. A módszernek viszont az a hátránya, hogy ront a teljesítményen, ami a felhasználói felület kirajzolásának és használatának akadozását eredményezi, ezért normál használatához érdemes az alkalmazásnak egy olyan fordítását használni, ahol a kiíratások kommentekbe lettek vonva.

Az MQTT hálózat brókereként a nagyobb, külső rendszer számára csoporttársam által megvalósított szervert használtam, ami képes az adatok naplózására, valamint a naplózott adatok kiküldésére, ezért a naplózott adatok lekérését is ennek segítségével teszteltem.

Az alkalmazásban a tesztelés során kreált modultól származó, beérkező üzeneteket egyrészt egy általam készített, az MQTT hálózatra felcsatlakozó, teszt adatokat generáló konzolos alkalmazást használtam, másrészt az egy-egy üzenettel tesztelő esetekhez, valamint az alkalmazás által publikált üzenetek megfigyeléséhez a Softblade cég MQTT.fx elnevezésű programját használtam.

7.2 Tesztelt körülmények, feladatok és folyamatok

Az alkalmazás bizonyos körülmények között a vészhelyzeti értesítésen kívül is küldhet lokális értesítést, a 7.1 ábrán ezeket a kiváltó körülmények tesztelését írtam le

Tesztelt körülmények	Elvárások	Eredmény
Hálózati kapcsolat szakadása.	Értesítés küldése.	Helyes működés
Hálózati kapcsolat felvétele.	Értesítés küldése.	Helyes működés
Modul kreálás vagy módosítás során felvitt modulazonosító már létezik, vagy rossz formátumú.	Értesítés küldése, mentés vagy módosítás leállítása.	Helyes működés

7.1. táblázat. Lokális értesítést előidéző körülmények tesztelése

Az alkalmazás fejlesztése alatt minden feladat megvalósítása során manuálisan teszteltem azok működését. A 7.2. táblázatban ismertetett tesztesetek során ezen feladatok összehangolt, adott esemény bekövetkezte általi működésük helyességét vizsgáltam.

A következő történések során lefutó folyamatok tesztelését írtam le. Az első az alkalmazás háttérbe vagy előtérbe kerülését követő folyamat, a második egy formátumhelyes üzenet érkezése után induló folyamat, a harmadik és negyedik pedig a modul vagy a tulajdonság táblák megváltozását követő folyamat.

1. teszt	Alkalmazás indítása, háttérbe küldése, aztán újra előtérbe hozása, azaz folytatása.
Elvárások	Kommunikációs egység alkalmazás indításakor vagy előtérbe kerülésekor csatlakozik a szerverhez és feliratkozik a témákra, háttérbe kerülésekor leiratkozik és bontja a szerverrel való kapcsolatot.
	Android platformon háttérbe kerülés során munkálatot beütemezteti, előtérbe kerülés közben lemondja.
Eredmény	Helyes működés
2. teszt	Üzenet beérkezett, kommunikációs egység felbontotta az üzenetet, és elindítja az üzenet beérkezését követő eseményt.
Elvárások	Adatbázis kezelő helyes formátumú adat esetén ment.
	Értesítéskezelő értesítés küldésre jogosult modul és helyes formátumú adat esetén kiértékeli, hogy szükséges-e értesítést küldeni. Amennyiben szükséges, akkor megteszi, és hamisra állítja a modul értesítésküldési tulajdonságát, majd elmenti a megváltozott modult.
	Nézetmodell frissíti a tulajdonság legutóbbi értékét, amennyiben egy modul tulajdonságait megjelenítő nézet van előtérben, és ha a megjelenített modul megegyezik azzal, amire az üzenet érkezett, valamint az adat formátuma is helyes.
Eredmény	Helyes működés
3. teszt	Adatbázis tulajdonság táblája változott, azaz egy vagy több tulajdonság lett mentve, módosítva vagy törölve.
Elvárások	Értesítéskezelő frissíti az értesítésre jogosult modulok és tulajdonságok listáját.
	Kommunikációs egység frissíti az MQTT feliratkozásokat.
Eredmény	Helyes működés

7.2. táblázat. Folyamatok tesztelése

Az utolsó nagy teszteset pedig az eddig tesztelt folyamatok kombinálásának is mondható, mivel a háttérmunkálatok során a felhasználói felület valamelyik nézetének frissítésén kívül közel mindegyik korábban leírt elvárásnak teljesülnie kell a háttérfunkciók helyes működéshez. Ez a teszt azt az esetet írja le, amikor az alkalmazás egy háttérmunkálatokat végző állapotában olyan üzenetet kap, ami alapján vészhelyzeti értesítést kell küldenie. Ekkor megváltoztatja az adott modul Tracking tulajdonságát is, hogy ne küldjön fölöslegesen több értesítést. Ennek a tesztnek a leírását a 7.3. táblázatban lehet olvasni.

4. teszt	Az alkalmazás háttérbe került, majd később, platformtól függő mennyiségű idő elteltével elindul a háttérmunkálat, miközben üzenetek érkeznek a témákra. Az üzenetek alapján az alkalmazásnak vészhelyzeti értesítést kellene küldenie.
Elvárások	Kommunikációs egység csatlakozik a szerverhez és feliratkozik a témákra, majd a háttérmunkálatok végével leiratkozik és bontja a szerverrel való kapcsolatot.
	Kommunikációs egység továbbá üzenet beérkezését követően felbontja az üzenetet, és elindítja az üzenet beérkezését követő eseményt.
	Adatbázis kezelő egység helyes formátumú adat esetén menti a mérést.
	Értesítéskezelő egység értesítés küldésére jogosult modul és helyes formátumú adat esetén kiértékeli, hogy szükséges-e küldeni. Szükség esetén megteszi, és hamisra állítja a modul Tracking tulajdonságát, majd elmenti a módosított modult.
	Adatbázis kezelő egység a megváltozott modul mentése után elindítja a modul tábla változását követő eseményt.
	Értesítéskezelő egység frissíti az értesítésre jogosult modulok és tulajdonságok listáját.
	Kommunikációs egység feliratkozik a témákra.
Eredmény	Helyes működés

7.3. táblázat. A háttérmunkálatok tesztelése

8 Eredmények és továbbfejlesztési lehetőségek

8.1 Megvalósítás és tesztelés eredményei

A megvalósítással és teszteléssel végezve a fejlesztési fázis eredménye egy olyan platformfüggetlen alkalmazás, ami Android, iOS és UWP platformokon is az elvárásokat kielégítő módon használható, és megoldja az eredeti problémafelvetés során leírt problémát. Ezen túl pedig teljesíti a tőle elvárt feladatokat, miközben a használata és vizuális felépítése kellően egyszerű maradt, hogy felhasználóbarát legyen elkészült állapotban is.

8.2 Továbbfejlesztési lehetőségek

8.2.1 QoS szintek állíthatósága

További fejlesztési lehetőségekhez felírható a feliratkozásokra, valamint a publikálási funkciókra vonatkozó, akár témánkénti QoS szint beállíthatósága. Ezzel a felhasználó kezébe kerülne az üzenetfolyam biztosításának személyre szabhatósága.

8.2.2 További optimalizálás

A teljesítmény és hatékonyság javításának érdekében a beérkező mérési adatok formátum ellenőrzését egy erre használt egység végezhetné, hogy az adatokat ne több osztálynak kelljen magának megvizsgálnia.

8.2.3 UWP háttérmunkálatok

Jelen esetben az alkalmazás UWP rendszereken futva nem végez periodikus háttérmunkálatokat, mivel asztali számítógépes használat során az alkalmazásnak nem kell előtérben tartózkodnia, hogy figyelni tudja az MQTT hálózatot. Ettől függetlenül viszont egy UWP alkalmazás futtatható az arra alkalmas tableteken, okostelefonokon, de akár még a Microsoft HoloLens-én, vagy az Xbox One konzolon is. Ilyen eszközökön már viszont ugyanannyira hasznos lenne háttérmunkálatok használata, mint az Android, vagy iOS verziókon, ezzel jó további fejlesztési opcióvá téve a háttérmunkálatok implementálását is.

9 Tartalmi összefoglaló

Szakdolgozatom megalkotása során a célom egy Android, iOS és UWP rendszereken használható platformfüggetlen alkalmazás fejlesztése volt egy távoli elérési és vezérlési funkciókkal ellátott forgógép védőmodulhoz. Az alkalmazás egyszerűen használható felületet biztosít a felhasználó számára, amin keresztül igénybe tudja venni a modul távoli elérési és vezérlési funkcióit, amennyiben internetkapcsolattal rendelkezik, és a rendszer adatfolyamáért felelős szerver is működésben van. Az alkalmazás célja a modul funkcióinak elérhetővé tétele, és a felhasználó által használt forgógépek, mint például szivattyúk aktuális és múltbéli állapotainak leolvasási módjának könnyítése úgy, hogy azt bárhonnan elvégezhetővé teszi.

A tervezés számomra az alapok lefektetésénél kezdődött. A kiindulópontot az alapvető feladatok leírása és a megoldandó probléma megfogalmazása jelentette. Utána irodalomkutatásba fogtam. Ennek során ismertettem az Internet of Things informatikai területet, valamint elterjedtségi mértékének múltját és jövőbe mutató irányvonalát. Bemutattam az MQTT kommunikációs protokoll történetét és működését, valamint megvizsgáltam, hogy a protokoll hogyan szabja meg a motorvédő modul kommunikációs hálózatát, amibe az alkalmazás beépül. Ezután a platformfüggetlen alkalmazás fejlesztés területéről írtam, ami során kitértem az ilyesféle fejlesztés céljaira, előnyeire és hátrányaira, valamint megvizsgáltam két ilyen fejlesztéshez használható keretrendszert is. Az irodalomkutatás utolsó fázisában pedig különféle értesítési rendszereket és használati eseteiket mutattam be, ami után pedig a kutatás során megismert, megvalósításhoz használható technológiák közül választottam, döntéseimet megérvelve.

A kutatás végeztével, az igényspecifikáció során már pontosabb módon tudtam megfogalmazni az alkalmazással szemben állított elvárásokat, aztán a rendszerterv létrehozásába fogtam, aminek során létrehoztam az alkalmazás architekturális tervét, valamint leírtam az egységek pontos feladatait és tervezett működését is.

Ezek után a megvalósításról és a tesztelésről írtam. A megvalósításról szóló fejezet során igyekeztem bemutatni a fontosabb, érdekesebb, vagy bonyolultabb részeit a fejlesztett alkalmazásnak, amelyeket megvalósításuk közben is teszteltem a tesztelésről szóló fejezetben leírt módszerek szerint, hogy minél hamarabb helyesen működő egységekkel tudjak előállni.

A megvalósítással végezve pedig az alkalmazás egységeinek együttműködésének helyességét megkövetelő nagy folyamatok és feladatok teszteléséhez, és ezek leírásához láttam. A szakdolgozatomat a fejlesztési fázis eredményeinek értékelésével, valamint az elkészült alkalmazás továbbfejlesztési lehetőségeinek ismertetésével zártam.

10 Summary

During the creation of my thesis, my goal was to develop a platform-independent application that can be used on Android, iOS and UWP systems for a rotary machine protection module with remote access and control functionalities. The application provides the user with an interface that is easy to handle and through which the user is able to draw on the module's remote access and controlling functions, given that they have internet connection and the server responsible for the system's data flow is also in operation. The application's goal is to make the module's functionality available and to make for example the reading of a suction-pump's current and past data easier by making it possible to do from anywhere.

The design for me began with laying the foundations. Writing about the basic tasks and the to be solved problem marked the starting point. After that, I began researching literature. Through that I presented the Internet of Things IT field, and the past and the future facing trend of its prevalence. I presented the history and the operation of the MQTT communication protocol, and I also examined how the protocol defines the motor protection module's communication network, which the application is incorporated into. After this I wrote about the field of platform-independent development, during which I covered the goals, the advantages and disadvantages of this kind of development, and I also examined two frameworks used for this type of development. In the last phase of the literature research, I presented various notification systems and their use cases, after which I chose out of the introduced technologies that could be used for the implementation and gave reasoning for my decisions.

After the completion of the literature research, I was able to outline the expectations set for the application more precisely in the requirement specification, and then I started the creation of the system design, in which I created the application's architectural plan, and I wrote down the exact tasks and the planned operations of the components.

Then I started implementing and testing. In the chapter about the implementation I strived to present the more important and interesting or complicated parts of the developed application, which parts I was also testing in ways I later explained during the chapter about testing while implementing them, so that I was able to create properly working components as soon as possible.

After finishing the implementation, I started testing and documenting said tests about the big processes and tasks which require the components to work together properly. I concluded my thesis with the results of the development phase and with the descriptions of further development possibilities.

11 Irodalomjegyzék

[1] Chakraborty, M.: IoT Standardization: Why IoT has not set standards yet?, (https://www.analyticsinsight.net/iot-standardization-why-iot-has-not-set-standards-yet/), utoljára megtekintve: 2022.04.29.

[2] List of articles citing 'Internet of Things (IoT): A Literature Review', (https://www.scirp.org/journal/papercitationdetails.aspx?paperid=56616&journalid=2431), utoljára megtekintve: 2022.04.29.

[3] Madakam, S., Ramaswamy, R., Tripathi, S.: Internet of Things (IoT): A Literature Review. Journal of Computer and Communications Vol. 3. (No. 5.), 2015, 164–173. old.

[4] Vailshery, L. S.: Global IoT and non-IoT connections 2010-2025, (https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/), utoljára megtekintve: 2022.04.29.

[5] Jovanovic, B.: Internet of Things statistics for 2022 - Taking Things Apart, (https://dataprot.net/statistics/iot-statistics/), utoljára megtekintve: 2022.04.29.

[6] Lombardi, M., Pascale, F., Santaniello, D.: Internet of Things: A General Overview between Architectures, Protocols and Applications. Information, Vol. 12. (No. 2.), 2021, 87. old.

[7] Babu, N. S. C., Dwarakanath, V. T., Haribabu, P., Singh, V. P.: IoT standardization efforts — An analysis. International Conference On Smart Technologies For Smart Nation (SmartTechCon), Bengaluru, India, 17-19. Augusztus 2017.

[8] Molnár, M., Váradi, T., Zdroba, D.: EGYFÁZISÚ FORGÓGÉP VÉDELMI ÉS TÁVVEZÉRLŐ MODUL. Óbudai Egyetemi 54. Tudományos Diákköri Konferencia, Budapest, Magyarország, 17. November 2021.

[9] Makwana, A., Soni, D.: A survey on mqtt: A protocol of internet of things (IoT). Proceedings of the International Conference on Telecommunication, Power Analysis and Computing Techniques (ICTPACT-2017), Chennai, India, 6-8. Április 2017.

[10] MQTT - The Standard for IoT Messaging, (https://mqtt.org/), utoljára megtekintve: 2022.04.29.

[11] Zein, S., Zohud, T.: Cross-Platform Mobile App Development in Industry: A Multiple Case-Study. International Journal of Computing, Vol. 20. (No. 1.), 2021, 46–54. old.

[12] Arya, H., Britch, D., Dunn, C., Johnson, J.: What is Xamarin?, (https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin), utoljára megtekintve: 2022.04.29.

[13] The Good and The Bad of Xamarin Mobile Development, (https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/), utoljára megtekintve: 2022.04.29.

[14] Işitan, M., Koklu, M.: Comparison and Evaluation of Cross Platform Mobile Application Development Tools. International Journal of Applied Mathematics Electronics and Computers, Vol. 8. (No. 4.), 2020, 273–281. old.

[15] What Is React Native?, (https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html), utoljára megtekintve: 2022.04.29.

[16] Fast Refresh, (https://reactnative.dev/docs/fast-refresh), utoljára megtekintve: 2022.05.08.

[17] Push Notifications Statistics, (https://www.businessofapps.com/research/push-notifications-statistics/), utoljára megtekintve: 2022.05.08.

[18] Langholz, S.: Understanding iOS Remote vs Local Push Notifications, (https://onesignal.com/blog/understanding-ios-remote-local-push-notifications/), utoljára megtekintve: 2022.05.08.

[19] Meads, A., Paniagua, C., Srirama, S.,Warren, I., Weerasinghe, T.: Push Notification Mechanisms for Pervasive Smartphone Applications. IEEE Pervasive Computing, Vol. 13. (No. 2.), 2014, 61-71. old.

[20] How push works, (https://web.dev/push-notifications-how-push-works/), utoljára megtekintve: 2022.05.08.

[21] Johnson, J., Britch, D., Coulter, D., Davis, O., Dunn, C.: Xamarin.Forms Local Databases, (https://learn.microsoft.com/en-us/xamarin/xamarin-forms/data-cloud/data/databases), utoljára megtekintve: 2022.11.21.

[22] Bilgin C.: Mobile Development with .NET: Build Cross-Platform Mobile Applications With Xamarin.Forms 5 And Asp.Net Core 5. *Packt Publishing*, 2021, ISBN 978-1-80020-469-0

[23] Johnson, J., Britch, D., Coulter, D.: Local notifications in Xamarin.Forms, (https://learn.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/local-notifications), utoljára megtekintve: 2022.11.21.

12 Ábrajegyzék

3.1. ábra. Aktív IoT és nem IoT kapcsolatok mennyisége az évek során 10
3.2. ábra. Tipikus felépítése egy MQTT hálózatnak 12
4.1. ábra. Az alkalmazás használati eset diagramja 21
4.2. ábra. Az alkalmazás tervezett menürendszere 21
5.1. ábra. A teljes kommunikációs hálózat 26
5.2. ábra. Az applikáció architektúrája egységekre bontva 26
5.3. ábra. Az adatbázis E-K diagramja 27
6.1. ábra. Xamarin a Visual Studio telepítő alkalmazásában 31
6.2. ábra. Az adatbázis konfigurációja 31
6.3. ábra. A listafelület UWP, Android és iOS platformokon 33
6.4. ábra. Modulfelvétel UWP, Android és iOS platformokon 34
6.5. ábra. Kontextus menü UWP, Android és iOS platformokon 34
6.6. ábra. Tulajdonságok nézete UWP, Android és iOS platformokon 35
6.7. ábra. Tulajdonság méréseinek listája UWP, Android és iOS platformokon 35
6.8. ábra. Tulajdonság szerkesztése UWP, Android és iOS platformokon 36
6.9. ábra. Oldal betöltésekor induló frissítés metódus hívási lánca 37
6.10. ábra. A figyelt tulajdonságok listájának lekérdezése 38

13 Táblázatjegyzék

3.1. táblázat. Platformfüggetlen fejlesztés előnyei és hátrányai 15
7.1. táblázat. Lokális értesítést előidéző körülmények tesztelése 41
7.2. táblázat. Folyamatok tesztelése 42
7.3. táblázat. A háttérmunkálatok tesztelése 43