ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK	Hallgató neve:	**Juhász Zoltán**
2022	Hallgató törzskönyvi száma:	T/004666/FI12904/N
	Neptun kódja:	[redacted]

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Juhász Zoltán**
Törzskönyvi száma: T/004666/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Automatizált mobilparkolás alkalmazás
Automatized mobile-parking application

Intézményi konzulens: Simon-Nagy Gabriella
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Számítógép architektúrák
Szoftverrendszerek fejlesztése specializáció

A feladat

Tervezzen és valósítson meg egy mobilalkalmazást, amely segíti a parkolás menedzselését a felhasználó aktuális pozíciója és korábbi tevékenységei alapján. Kutassa fel és vizsgálja meg a feladat megoldásához használható technológiákat. Ismertesse a hasonló célú alkalmazások működését és képességeit. Valósítson meg egy olyan mintaalkalmazást, amely a felhasználó aktuális pozícióját és korábbi viselkedését és preferenciáit figyelembe véve a parkolás kezdetekor megtalálja az aktuális parkolási zónát, menedzseli a megfelelő összegű díj befizetését, és figyelmeztet a parkolási idő lejártára, valamint az esetleges hosszabbításra.

A dolgozatnak tartalmaznia kell:

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

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

P.H.

.....
Dr. Vámossy Zoltán
intézetigazgató

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

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat 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. 05. 15.

.....

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Juhász Zoltán

Neptun Kód: [redacted]

Tagozat: Nappali

Telefon:

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

Szakdolgozat címe magyarul:
Automatizált mobilparkolás alkalmazás

Szakdolgozat címe angolul:
Automatized mobile-parking application

Intézményi konzulens:
SIMON-NAGY GABRIELLA

Külső konzulens:

Alk.	Dátum	Tartalom	Aláírás
1.	2021. 09. 21.	A téma meghatározása	
2.	2021. 10. 05.	A kutatási területek áttekintése	
3.	2021. 10. 19.	Az irodalomkutatás összegzése	
4.	2021. 11. 09.	A rendszerspecifikáció véglegesítése	

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

.....

Intézményi konzulens

Budapest, 2021. november 9.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Juhász Zoltán

Neptun Kód: [redacted]

Tagozat: Nappali

Telefon: [redacted]

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

Szakdolgozat címe magyarul:

Automatizált mobilparkolás alkalmazás

Szakdolgozat címe angolul:

Automatized mobile parking application

Intézményi konzulens:

SIMON-NAGY GABRIELLA

Külső konzulens:

Alk.	Dátum	Tartalom	Aláírás
1.	2021. 11. 22.	Megvalósítási lehetőségek áttekintése	
2.	2021. 12. 02.	Fejlesztési fázis ellenőrzése	
3.	2021. 12. 10.	Tesztelés ellenőrzése	
4.	2021. 12. 14.	Eredmények, összegzés	

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

.....

Intézményi konzulens

Budapest, 2021. december 14.

TARTALOMJEGYZÉK

1 BEVEZETŐ

Az ipariforradalom, a városok növekedése továbbá az egyéni közlekedés elterjedésének következményeképp Európában - ahol a városok többsége a középkor óta több százéves kis szűk utcákkal üzemelnek, ideértve Magyarországot azon belül Budapesten - a parkolóhelyek és a forgalomban lévő autók aránya ez utóbbiak felé oly mértékben eltolódott, hogy az addigi ingyenes parkolási gyakorlatot felváltotta a fizetős parkolás. A probléma az ingyenes időkorlát nélküli parkolással az volt, hogy az autósok sokáig használták a helyet, időnként aránytalanul sokáig, ezáltal akadályozva a mindennapi teendőkben a többi városlakót, akiknek szükségük lett volna parkolóhelyekre ideiglenesen. A fizetős parkolással, ezzel együtt a parkolási idő korlátozásával ezt a problémát szerették volna megoldani a városvezetések szerte a kontinensen az 1970-es és 80-as években bevezetett fizetős parkolás rendszerével.

A dolgozatnak nem célja tárgyalni, hogy ezen rendszerek, megoldották-e a parkolás problémáit.

A kezdeti rendszerek az utcákon elhelyezett, mindannyiunk által ismert nyomtatós fémpénz bedobós szerkezetek voltak, amelyeket a technológia fejlődése folyamán a 2000-es évek elején kiegészített az akkoriban elterjedőben lévő mobiltelefonok által SMS-ben elindított parkolás. 2010 körül további könnyítés érdekében megjelentek a mobilalkalmazások, amelyekkel már nem kellett zónakódokat beütni, egyszerűen használhatóak voltak – bár funkciójukban néha még akadtak hiányosságok.

Ezen rendszerek mindegyikében közös volt a felhasználó aktív részvétele a fizetési folyamat elindításában mind az automatába pénz bedobás, mind az SMS küldés, zónakóddal, rendszámmal, illetve az applikációs mobilparkolás esetében. Ez utóbbi kettő igényelte/igényli a felhasználó aktív részvételét a parkolás leállításához, amennyiben nem fix időtartamot jelölt meg az erre alkalmas rendszereken.

Az önkormányzatok fizetős parkolási zónákat jelölnek ki, ahol kötelező a várakozási díj megfizetése. Ennek elmulasztása pénzbüntetéssel jár. Továbbá az új kényelmi lehetőségek, SMS parkolás, applikációs parkolás esetében nem csupán az elindítás elmulasztása esetén járhatunk rosszul, hanem a leállítás elfedése esetén is túlfizethetünk, feleslegesen.

Ezen problémák megoldására készíteném el szakdolgozatomban azt az applikációt, amely egyszer s mindenkorra megoldja a fizetős parkolásból eredő büntetéseket, túlfizetéseket, azzal, hogy a fizetés elindítását teljesen automatikusan végre hajtja, ezzel elejét veszi a parkolási büntetéseknek, illetve a fizetés leállítását is amellyel a túlfizetésből adódó veszteséget szűnteti meg.

1.1 A probléma fontossága, felvezetése

Jelenlegi felgyorsult világunkban számtalan olyan helyzetbe kerülhetünk, ahol a mindennapi rohanás hevében elfelejtjük megfizetni a pakolás díjját. Ez jobb esetben akkor jut eszünkbe amikor még benne vagyunk a türelmi időben (átlagosan 5 perc) és még van esélyünk vagy visszaszaladni a járművünkhöz vagy távolról elindítani a mobilparkolást SMS-ben vagy applikációval. Amennyiben ez az idő letelt, már csak a szerencsében bízhatunk, hogy addig nem jött a parokolóőr – ennek esős időben nagyobb a valószínűsége.

Ez a plusz stressz faktor senkinek az életéből nem hiányzik, továbbá a parkolási céghez való befáradás idő, az ott befizetés egy olyan megaláztatás amire senkinek nincs szüksége. Mindez egy egyszerű elfelejtés eredménye, ami mindannyiukkal megtörténhet.

Ezen stresszfaktorok életünkből van kiiktatására, továbbá a pénzbüntetések, túlfizetések elkerülésére érdekében készítem el jelen szakdolgozatban megjelölt applikációt.

1.2 A feladat pontos ismertetése

Az applikáció legfontosabb funkciója a parkolás elindítás és leállítás jelzése. Kétféleképp tervezek elindulni a megoldáshoz vezető úton, az egyik egy teljesen szoftveres a másik megoldás hardvert is igényel.

Az applikációnak az autó lezárásától vagy a felhasználó „mozgásmód” megváltozásától számított 5 percen belül elindítja, nyitás esetén pedig azonnal vagy beállítástól függően itt is 5 percen belül leállítja a parkolás.

A hardverelem nélküli megoldás a telefon GPS információi alapján mozgási sebesség meghatározással a gyorsulásmérő adatait felhasználva ismerné fel az autóról gyaloglásra váltás, az autó saját Bluetooth rendszerére (amennyiben létezik) való csatlakozással és lecsatlakozással, előre beállítható lakcím, és célcím megadással, a szokásos parkolási helyek eltárolásával továbbá a területen érvényes parkolási zónák figyelembevételével indítaná el vagy állítaná le a parkolást automatikusan vagy értesítések segítségével, amelyben a felhasználó állíthatná be, hogy mi legyen az alapértelmezett reakciója a rendszernek, amennyiben figyelmen kívül hagyja az értesítést. A leállításhoz az autó hozzávetőleges helyzetére való visszatérést, illetve ismét a mozgási mód megváltozását venné figyelembe.

A hardveres megoldás esetén azt az információt használjuk fel, hogy a kocsi állapota a kulcs jelére lezártra/kinyitottra változik ezt érzékelve eszköz által Bluetooth kapcsolat segítségével tájékoztatja a telefonon a háttérben futó alkalmazást a lezárás és kinyitás tényéről, amely a területi parkolási zónák és időkorlátozások figyelembevételével elindítja vagy leállítja a parkolást.

2 IRODALOMKUTATÁS

2.1 Jelenleg a piacon lévő hasonló megoldások

Jelenleg sok szolgáltató van a piacon, akik ajánlanak megoldást a problémára. Ezek azonban rendre beépítendő eszközöket igényelnek, továbbá havi díjas előfizetést, mert szolgáltatásukat egybecsomagolva árulják, különböző funkciókkal egybekötve, mint például GPS flottakövetés, sofőrazonosítás stb. Így szolgáltatásaikat leginkább cégeknek éri meg megvásárolni. Egyéni felhasználóknak értelmetlen lenne beruházni a több tízezer forintos, amely adott esetben több, mint százezerforintos központi egységre és ezen felül fizetni havidíjat, csupán csak egy kényelmi funkcióért. A bekerülési költség soha nem fog megtérülni. Így hiába hasznos a funkció, ha az átlagos büntetési tétel értékének sokszorosát kell kifizetni érte. Azontúl a beszerelés miatt az autónkról is lemondhatunk 3-4 órára, ami miatt külön szabadságot is lehet ki kell venni.

Magyarországon relatív sok szolgáltató kínál hasonló megoldásokat a piacon ilyenek például az Obu City – skyguard és az iData – iTrack valamint az ITS Protection – ITS GO! termékek. Az összes hazai szolgáltató megoldásai valamilyen módosításokat igényelnek a járművön mely kisebb-nagyobb kényelmetlenséggel jár és amelyek költség vonzata változó nagyságú ugyan, de reális megtérülési idejük átlagos felhasználó számára sok éves lehet és az általános havidíj miatt nem is biztos, hogy befektetésük valaha is megtérül. Közös bennük, hogy a parkolásra, mint plusz szolgáltatásra, szolgáltatás kiegészítésre gondolnak, így havidíjaik magasak.

Látható tehát hogy reális alternatívát nem kínál egyik cég se, pedig a probléma valós, ráadásul megoldása relatív egyszerűen kivitelezhető. Mindegyik a céges vásárlókat próbálja elérni, így nem is fejlesztenek a széles tömegeket irányába. A magas áraik miatt kizárják a potenciális felhasználók nagy részét, akik akár még fizetnének is egy jóval reálisabb árat a szolgáltatásért.

Szakdolgozat ötletem kitalálásakor nem volt tudomásom jelen cégek létezéséről és szolgáltatásaik mibenlétéről, pusztán én is egyike voltam azon személyeknek, akiket a feledékenység átka utolért.

Célom így jóval egyszerűbb megoldás összeállítása, mely bár szolgáltatásait tekintve a parkolási díj megfizetésére szorítkozik, de azt beszerelésében, használatában is leegyszerűsíti, hogy mindenki számára elérhető és hozzáférhető legyen a stresszmentes parkolás.

2.2 Hardveres segítség az applikáció számára

Ezen megközelítési mód esetében az elindításhoz és a leállításhoz kapcsolatot kell teremteni az applikáció (a telefon) és a jármű valamely oly eleme közt, amely tájékoztatást tud adni az autó állapotáról. Ehhez külső eszközt kell felhasználnunk. Előnyben részesíteném ezt a megoldási típust mivel ezen megvalósítási forma kétséget kizáróan jelezni tudja az applikáció számára a parkolásindítás igényét, leállítását, egyszerűen a jármű kulcsának „gombnyomására". Lecsökkentené a szoftver méretét és még nagyobb platform függetlenséget adna.

2.2.1 A rádiófrekvenciás kulcs

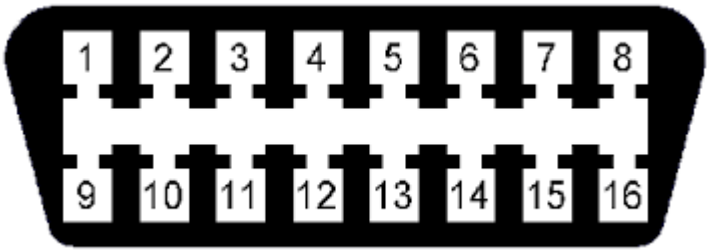
A lezárt állapot elindítását a legtöbb járműben rádiófrekvenciás kulcs szabályozza, amely jelet küld az autó elektronikájának, amely elindítja a lezárási folyamatot. Ez a jelet elektromágneses sugárzás formájában Európában a 433,92 Mhz-es tartományban sugározzák a kulcsok. Bár a legtöbb mai okostelefon rendelkezik rádióantennával, ezek vételi frekvenciája sajnos kívül esik ebből a tartományból. Ezen túl a kulcs jelét nem lehet egyszerűen olvasni. Biztonsági okokból kifolyólag, ezeket a 1990-es évek óta ellátják védelemmel, amely pszeudorandom számgenerátorral működik, amelyből kettő azonos kezdő értékkel ellátott eszköz van, egy a kulcsban, egy a járműben elhelyezett vevőegységben. Ennek következő értékét a kulcs, mint „nyitási jelet" küldi el és a vevő amennyiben a megadott frekvencián érzékeli ezt a pszeudorandom számot és amennyiben ez megegyezik az ő szerinte következővel kinyitja, avagy lezárja a járművet.

Azt biztonsággal megállapítani, hogy a saját kulcsunk küldte-e a jelet a rendszer tervezéséből fakadóan nem lehetséges egy harmadik eszköz számára. Ezért a kulcsot és a járműben található vevőegységet le kellene cserélni egy olyan eszköz hármasra, amely az új kulcsból, vevőegységből és a telefonra jeltovábbító eszközökből állna. Ez számomra gyakorlatilag kivitelezhetetlen minden járműre.

2.2.2 Az OBD csatlakozó és a CAN-BUS rendszer, valamint kiolvasásuk

Az Európai Unió területén 2001-től az összes benzines 2004-től az összes gázolajjal működő újonnan forgalomba állított járművön kötelező elhelyezni ezt a diagnosztikai csatlakozót. A CAN busz rendszert a Bosch fejlesztette ki 1983-ban az autóban található rendszerek egymás közötti kommunikációját biztosítandó.

Az irodalomkutatásom elején az volt a tervem, hogy az OBD szabvány keretein belül kérdezném le a járműtől a lezárására vonatkozó információkat, de sajnos ez nem lehetséges, mivel az OBD szabványban csak olyan diagnosztikai célú olvasási és írási műveleteket lehet végezni, ami a motor működését érinti. Ekkor figyelmem a CAN-BUS rendszerre és annak megismerésére irányult át.



2.1. ábra, Az OBD csatlakozó kiosztása

A 2.1. ábra-n látszik egy OBD csatlakozó ezek lábainak a számunkra érdekes kiosztásai a 6-os és a 14-es láb amely a CAN HIGH(6) és a CAN LOW(14).

Ezen rendszer része az OBD-nek (6,14-es lábak) és egy bizonyos parancssor kiadása után olvashatóvá is válik az egész busz, mint adatfolyam. A manapság kapható ELM327 Bluetooth OBD kiolvasók mindegyike konfigurálható úgy, hogy kiolvasható legyen a CAN busz.

A CAN busz kiolvasásához a következő utasításokat kell elküldeni az eszköznek pontosan ebben a sorrendben

```
//ELM327 eszköz visszaállítása
ATZ
//Alapbeállítások beállítása
ATD
//A visszajelzés kikapcsolása
ATE0
//A CAN protokoll SAE J1939-ra kapcsolása
ATSPB
//A szóközök kikapcsolása
ATS0
//Soremelés kikapcsolása
ATL0
//Hosszú üzenetek engedélyezése
ATAL
//Néhány ELM327 eszköznél segíti a kiolvasást
ATPBC001
//Fejléc bekapcsolása
ATH1
//A CAN busz monitorozásának megkezdése
ATMA[1]
```

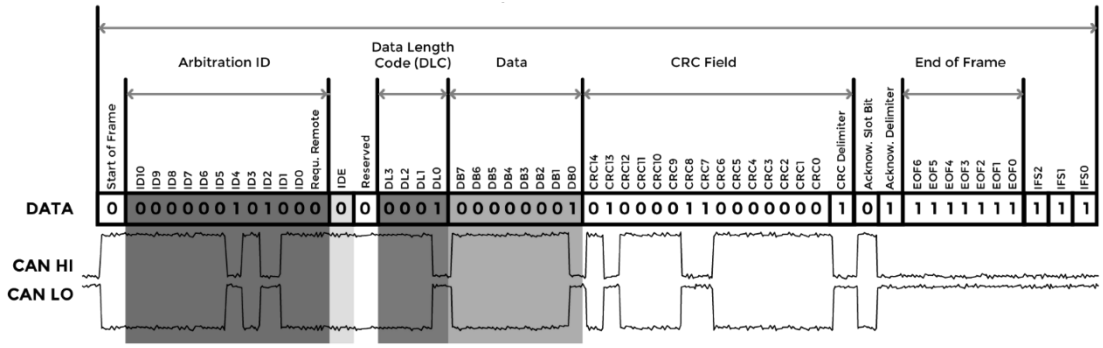
Az ilyen módon konfigurált ELM327 Bluetooth eszköz elkezdi elküldeni a CAN buszon található adatokat hexadecimális formátumban.

Ezeknek az eszközöknek a rendelkezésre álló memóriájuk véges és hamar betelik. Ilyenkor időről-időre el kell küldeni az ATMA parancsot a buffer törléséhez, és az adatfolyam monitorozás újraindításához.

2.2.3 Az nyers CAN adatok dekódolása

Az eszköztől megkapott hexadecimális kódokat szét lehet választani azonosítóra és üzenetekre.

A CAN busz azonosítók gyártóról gyártóra változnak, beazonosításukhoz szükség van tesztelésre. A CAN busz rendkívül „zajos”, sok járműrész kommunikál rajta ezért ki kell



2.2. ábra, A CAN BUS jeleinek értelmezése

szűrnünk belőle a jármű nyitásának azonosítóját.

Gyakorlatilag intervallum felezés segítségével eldöntjük, hogy a rögzített kódsorok melyike tartalmazza, a kulcs nyitás azonosítóját. Először is végre kell hajtani az eseményt - megnyomni a gombot. Ezalatt a beérkezett üzenetek el kell tárolni. majd részletenként vissza kell játszani a jármű számára és megfigyelni a jármű reakcióját. Amennyiben a kívánt akciót elértük, megfelezzük a visszajátszott azonosítók részét és tovább folytatjuk ezt a felezést egészen addig amíg a jármű kinyílt és a visszajátszott azonosítók száma elérte az egyet. Ezt az azonosítót feljegyezzük, így kiderülhet számunkra az mely azonosító kód tartalmazza a járműnyitás funkciót.

Számunkra ezen információ „dekódolása” elegendő az alkalmazás funkcióinak ellátásához.

2.2.4 A vezetéknélküli kiolvasás biztonsági kérdései

A CAN busz mivel a jármű belső elektronikai komponensei között teremt kapcsolatot tervezéséből fakadóan nem védett semmilyen támadás kapcsán. Mint az előző bekezdésben tárgyaltuk a CAN buszon az elfogott üzeneteket vissza is lehet játszani a jármű számra ezzel utasításokat is lehet adni a járműnek. Sajnálatos módon az általánosan hozzáférhető ELM327-es Bluetooth OBD olvasóknak könnyedén hozzáférhető a

Bluetooth kapcsolathoz szükséges csatlakozási kódjaik, amelyek általanosan a készülékek 90%-án az 1234-es kódot használják.

Ahogy mi megfejtettük „reverse engineering” segítségével eme biztonságilag kritikus pont kódját, már csak keresnünk kell egy hasonló járművet, amiben szintén benne van az ELM327 majd a Bluetooth rendszer egyszerű kódját feltörve lejátszani a járműnek a kulcs nyitás azonosítóját és üzenetét és máris el lehetne tulajdonítani azt, megkerülve annak biztonságtechnikai kialakításait. Bár ennek kicsi a valószínűsége, de mivel a járműveink egyaránt nagy használati és pénzbéli értéket képviselnek, ezért a kényelem oltárán feláldozni a biztonságukat nem megengedhető.

Tulajdonképpen tudtunkon kívül egy „back door” -t, azaz hátsó ajtót hoznánk létre a járművünk számára, amelyet egy megfelelő információkkal rendelkező tolvaj kihasználva eltulajdoníthatná a járművet.

2.2.5 A hardveres megoldás konklúziója

Látható tehát, hogy OBD csatlakozón keresztül, ELM-327-es eszközzel lehetséges a CAN busz adatok folyamatos monitorozása majd Bluetooth kapcsolat segítségével ezen információk a telefonra továbbítása további feldolgozásra és dekódolásra mely információkkal lehetséges ezek után a parkolás elindítása és leállítása 100%-os pontossággal. Mindazonáltal ezzel egy biztonsági rést ütnénk a jármű egy védtelen kulcsfontosságú részén kitéve azt a lopás kockázatának, megkerülve annak riasztóját. A kényelmesség nem mehet a biztonság rovására.

Az OBD csatlakozóra való rácsatlakozás bár a végfelhasználók többsége számára nem okozna ugyan problémát, de szignifikánsan megnehezítené az alkalmazás üzembeállításának folyamatát, mivel eredetileg annak szántam az alkalmazást, hogy a lehető legkevesebb módosítással a lehető legpontosabban határozza meg a parkolási szándékot. Ráadásul azon felhasználók részére, akik járművében az OBD csatlakozó olyan helyre került kialakításra, hogy az nehezen hozzáférhető legyen, (biztonsági okokból) nekik ezen megoldás használatához szintén egy szervízbe való költséges látogatás lenne az egyetlen járható út.

A fentiekből következően a parkolás hardveres indításának forrása mind a kulcs jelének olvasása mind az OBD CAN busz olvasása részéről technikai és biztonsági problémákba ütköznek. Megoldásukra egyszerű mindenki által hozzáférhető biztonságos módszer nem létezik a piacon. Ezért a következőkben a szoftveres megvalósításra koncentrálok.

2.3 A szoftveres megoldás

A szoftveres megoldás számára nehezebb – ha nem lehetetlen - megállapítani 100%-os biztonsággal a parkolásindítás/leállítás igényét. Ezen megoldás során indirekt módon felhasználom a telefonban található szenzorokat és a mobiloperációs rendszerek beépített mozgáskövető funkcióit, továbbá amennyiben létezik az autó bluetooth kihangosító rendszerére való automatikus/manuális le/fel csatlakozást, illetve utólag az autóban elhelyezhető Bluetooth jeladókat.

2.3.1 A mobil operációs rendszerek áttekintése

A szenzorok és szolgáltatások továbbiakban való pontosabb kifejtése érdekében először operációsrendszert kell választani. Sajnos, nem rendelkezek az Apple iOS rendszerre való fejlesztéshez szükséges eszközparkkal így kizárásos alapon az Android rendszer maradna bárminemű összehasonlítás nélkül, ellenben mivel iOS rendszerű a személyes telefonom és szeretném használni az itt kifejlesztett applikációt ezért más megoldás után nézve azt találtam, hogy léteznek cross-platform megoldások, amelyekkel lehetséges iOS applikációt létrehozni Mac rendszerű számítógép és azon futatott Xcode nélkül is.

2.3.2 Cross-Platform eszközök, Xamarin vs Flutter vs React Native

Ezen megoldások előnye a tradicionális natív mobilalkalmazás fejlesztésessel szemben, hogy egyetlen verziót kell elkészítenünk és azt mindkét platformra lefordítja, ezzel munkát, időt spórolva. A kereszt platform további előnyei közé tartozik, hogy az egyszer kódolt alkalmazás sokkal jobban karbantarthatóbbá válik, illetve könnyebb egységes mégis az adott platform sajátoságait figyelembe vevő alkalmazásokat készíteni.

Három nagy tech óriás fejlesztette ki/birtokolja ezeket a technológiákat, a Xamarin-t a Microsoft, a Fluttert a Google, a React Native-t a Facebook fejlesztette ki.

Az egyetemünkön a teljesített tárgyak során az alábbi fejlesztőkörnyezetekkel, illetve nyelvekkel kerültem kapcsolatba, így ezeket megszerezett ismereteim alapján hozom meg döntésem.

- Java és Android Studio
- C# és Visual Studio

Bár mind a React Native és Flutter is képes Visual Studio Code alatti működésre, fejlesztési nyelvükben eltérnek az általam ismert nyelvektől, amely a Flutter esetében a Dart, és a React Native esetében a Java Script.

Az alkalmazások natív verzióihoz képest sebessége tekintetében a Flutter számít a leggyorsabbnak, majd ezt követi a React Native majd a sor végén a Xamarin áll. Ezen eredményeket szinte az összes általam talált forrás alátámasztja[2]. Bár a natív verziókhoz képest mindegyik keresztplatformos teljesítmény hátránnyal és ebből fakadóan futási

sebesség csökkenéssel jár, és ezek közül a Xamarin teljesít a legrosszabbul, de ez nem olyan mértékű, hogy egy olyan könnyű súlyú alkalmazás, amely amúgy is többnyire a háttérben fut és tulajdonképpen eseményekre való feliratkozás során a visszahívásokra reagál, nem szempont az adott nyelv grafikai teljesítményének relatív hátrányossága.

Jelenleg az egyik legelterjedtebb kereszt-platformos fejlesztőkörnyezet a Xamarin[3] melynek nyelve a C#, amelyet egyetemünkön is tanulunk. A C#-ban és a Visual Studio-ban való jártasságom miatt, valamit az elkészítendő alkalmazás relatív alacsony grafikai súlya miatt a választásom a Xamarin-ra esett.

A Xamarin a már előbb említett teljesítménybeli hátránya mellé társul még egy pár specifikus problémája is. Például:

- Nagyobb fájl méretek, a debuggolás alatt, valamint a végső alkalmazásban is
- A kódban platform specifikus részek alkalmazása mindkét operációs rendszernek 1-1 külön szekcióban
- Relatív későn érkeznek meg a Xamarinhoz az egyes platformok új képességei

Az általam használni kíván Apple által nyújtott funkciók, amelyeket a Core Motion Framework részeként az iOS már a 7.0-ás főverziótól kiegészített Motion Activity Manager tartalmaz és amelyek használhatóak a natív környezetben (Xcode Objective-C) fejlesztett alkalmazások számára, addig ugyan ezek a funkciókat a Xamarin iOS SDK 12-es verziója tette elérhetővé majd 5 évvel később.

Az általam használni kívánt funkciók megkövetelik azt, hogy különbséget tudjak tenni a mobiltelefon felhasználójának mozgás állapotai között. Ezen eseményekben való megváltozás érzékelését a két platform külön-külön valósítja meg és ezek elérését platform specifikus kódrészletek létrehozásával tudom csak megoldani.[4]

A nagyobb fájlméretek által okozott probléma inkább a tesztelés során válnak majd problémává, mert a folyamatos újra fordítás és tesztelés során sokszor kell majd újra és újra feltölteni az alkalmazásokat a telefonokra, amely időigényesebbé teszi a folyamatot.

2.3.3 Közelségalapú eszközök a Beacon-ök (iBeacon, Eddystone)

Ezen eszközök Bluetooth LE-t használva sugározzák azonosítójukat beállítható időközönként és erősséggel. Fontos megemlíteni, hogy ezeket az eszközöket eredetileg nem erre a megoldásra szánták, hogy a felhasználó saját maga számára hozza őket létre, majd használja azt egy előre meghatározott módon. A Google Android és Apple iOS rendszerekhez kifejlesztett Eddystone és iBeacon egy olyan felületet biztosít, ami felhő alapokon előre konfigurálandó, melyekből a jeladó közelbe való kerülése esetén a telefon le tudja kérni az adott jeladóhoz kapcsolódó információkat, mint a telepítési helye, kapcsolódó hirdetések, weboldalak és azok funkciói, alkalmazások és azok funkciói, érdekességek, járat információk, pontosabb geolokáció, betéri navigáció könnyítés.

Ezen eszközök rendeltetésszerű használata igényli az előre konfigurálást, amely jelen esetben értelmetlen lenne, hiszen minden alkalmazásnak egy vagy több saját nem fixen helyzet kötött egyéb információkat nem tartalmazó egyszerű időszakos jeladásra képes eszközre van szüksége, amely pusztán létezésének tényével azonosítja az adott járművet, és hatókörbe való be-ki lépéssel tájékoztatja az alkalmazást a környezetről.

Fontos, hogy ezen eszközök többnyire egyirányú kommunikációt valósítanak meg és teljesen az adott alkalmazástól függ, hogy mihez kezd a beacon információval.

2.3.4 Az Eddystone „halála” és az iBeacon Androiddal való kapcsolódása az Android AltBeacon Library, valamint az Universal Beacon Library NuGet csomagok segítségével

Eleinte minden platformhoz a saját beacon technológiáját terveztem használni, viszont a Google 2021 április 1.-vel megszakította a támogatását az Google Beacon Platformnak és ezzel együtt az Eddystone alapú beacon eszközöknek, így kénytelen vagyok használni az egyetlen támogatásában megmaradt platformot és egy ahhoz tartozó API-t az Android AltBeacon Library-t, melynek létezik Visual Studio Xamarin-ban található NuGet package-val telepíthető csomagja amelyet beleépítve az alkalmazásba használhatóvá válik a Android alatt az Apple iBeacon eszközei. Így a továbbiakban kizárólag az iBeacon használatával, tulajdonságainak bemutatásával és Android alatti használatával fogok foglalkozni.

Módosult terveim szerint ugyan az Android Beacon Library Xamarin-nal kompatibilis verzióját használtam volna, de sajnos ennek a NuGet csomagnak a legutolsó kiadása 2015 szeptemberében volt és amelynek támogatottsága legalább 2 éve lejárt és már nem kompatibilis a ma használt elemekkel.

Más alternatíva utána nézve találtam egy másik NuGet csomagot, amelyet kipróbáltam még a fejlesztés előtt Visual Studio-ban egy gyors tesztalkalmazás létrehozásával, hogy egyáltalán kompatibilis-e a jelenlegi rendszerekkel, illetve, hogy képes-e detektálni Android eszközre feltelepítve az iBeacon eszköz jelenlétét. Azt tapasztaltam, hogy az Universal Beacon Library legutolsó stabil (2018-as) kiadása alkalmas ezen feladat ellátáshoz.[5]

2.3.5 Az iBeacon

Egyetlen nagy cég által üzemeltetett és piacon maradt beacon technológiaként az Apple iBeacon eszközeinek Androidon való használatához igénybe kell vennem az Universal Beacon Library NuGet csomagját. Ez tulajdonképpen le is egyszerűsíti a fejlesztést hiszen egyetlen féle képen kell csak konfigurálni a Bluetooth LE jeladókat, amely teljesíti az iBeacon kritériumait és nem szükséges az éppen aktuális platformra jellemző módon különbséget tenni az applikációban a konfigurálások tekintetében ezzel is csökkentve a platformspecifikus kódok igényét.

A beacon-ök iBeacon szabványban való konfigurálásához megkövetel az eszközök számára egy többszintű azonosító elhelyezését az arra alkalmas eszközökön. A legfelső szinten áll az UUID ez alatt található a Major és Minor értékek. Ezen értékek a szabványban való létezésének megértéséhez muszáj visszautaljak az előző bekezdésben foglaltakra miszerint, ezt a technológiát hirdetési céllal hozták létre bevásárló központokban való tájékozódás az adott boltokban való kisebb egységek megtalálásához. Így egy adott üzletlánc meg tud vásárolni számtalan eszközt és azokat egy UUID alatt tudja konfigurálni, amely azonosítja az eszközt, mint sajátját.

Az UUID egy rövidítés melynek kifejtése: Universally Unique Identifier az az Egyetemesen Egyedi Azonosító, amely 32 hexadecimális értékből áll, 5 csoportra bontva kötőjellel elválasztva. A Major értékeket az egyes üzletei közötti különbségtétel számára 0 és 65535 között, illetve ugyan ez igaz a Minor értékre is, amely már az adott üzleten belüli elhelyezett konkrét jeladóra vonatkozik. [6]

Az UUID első tag 8 darab hexadecimális számjegyből, másodiktól negyedik tagjáig 4 számjegyből, illetve az ötödik tagja, amely 12 számjegyből áll, így jön ki a 32 számjegy.

Például: ceca39db-8d46-4378-8ced-0b1e2fcc663d

Ezeket az értékeket két helyre kell megadnunk elméletileg. Egyik helye magán az iBeacon eszközön található, amelyet a jeladó sugároz, a másik egy központi adatbázis, amelyhez vétel esetén az adott telefon csatlakozva plusz információkat tudhat meg a iBeacon-hez kapcsolódóan.

Az alkalmazás számára az iBeacon-öket nem szükséges regisztrálni egy központi felületen ahhoz, hogy használni tudjuk őket, mivel semmi más információt nem szükséges hozzá társítani, csak azt, hogy egy már előre az alkalmazásba beregisztrált eszköz hatósugarában vagyunk. Az iBeacon eszközt ezért csak el kell látni egy egyedi UUID-val, amelyhez egy külső szolgáltató[7] weboldalát tervezem használni pusztán azért, mert gyors és kényelmes módja egy egyedi azonosító létrehozásának. Az azon a weboldalon elhelyezett „span.courier.green” azonosítóhoz tartozó értéket a weboldal minden frissítésre újra generálja. Ezt felhasználva szinte korlátlan számban lehet létrehozni ilyen 32 byte hosszúságú azonosítókat, amelyeknek ütközésének valószínűsége rendkívül alacsony.

Elméletileg a rendeltetés szerű használat szerint (ami a reklámozás) lehetséges lenne egyazon UUID alatt regisztrálni az összes iBeacon készüléket, amelyet az alkalmazás példányai használni fognak, de ebben az esetben biztosítani kéne, hogy a Major és Minor értékek soha ne egyezzenek egyetlen felhasználó telefonján található alkalmazásban se. Ez elméletileg egy 65535^2 számú keretet ad számomra az egyazon UUID alatt lévő potenciális iBeacon azonosítók kiosztására. Bár ez megkövetelne egy szervert az alkalmazások számára, amely kiosztja ezeket az azonosítókat az egyes alkalmazás példányok számára.

Eredeti terveim szerint az alkalmazást „stand-alone” formában szeretném létrehozni, amely nem követel meg semmilyen szervert, ennek minden előnyével és hátrányával együtt. Ugyanis az alkalmazások kiszolgálásához fenntartani egy szervert jelenleg meghaladja a lehetőségeimet.

Az így létrehozott letöltött azonosítót aztán el kell menteni az alkalmazásba és az iBeacon eszközre egyaránt. Az alkalmazás amikor a telefon érzékeli az ilyen UUID-vel ellátott iBeacon jelenlétét értesítést küld az alkalmazás részére, amely ennek és az adott helyinformációk felhasználásával eldönti, hogy elindítja/leállítja a parkolást.

2.3.6 Az beacon-ök jelenlegi problémái és konfigurálásuk

Sajnálatos módon irodalomkutatásom során nem találtam egyetlen egy olyan iBeacon kompatibilis beacon gyártót, amely szabadon rendelkezésre bocsátana olyan SDK-t vagy API-t, amellyel lehetséges lenne egy megvásárolt eszköz konfigurálása a fejlesztendő parkolás alkalmazásomból.

Mivel a beacon technológia fő területe a marketing a fizikai tér információt felhasználva ezért minden beacon gyártó készít egy saját alkalmazást, amivel és csak azzal lehetséges előre konfigurálni a gyártó jeladóit. Egyes gyártók az eszközei konfigurálásához megkövetel díjfizetést is.

Látható tehát hogy mivel a beacon technológia fő területe a marketing ezért a cégeknek egyáltalán nem áll érdekében rendelkezésünkre bocsátani programkódba beépíthető konfigurálási függvényeket, hogy a végfelhasználó egy a tőlük különböző alkalmazásban tudná beállítani a Bluetooth jeladót, hogy egy teljesen más célra tudja használni az eszközt. Véleményem szerint ezzel korlátozzák a technológiájuk nagyobb elterjedését így végső soron saját bevételüket is. Hány és hány esetben tudná megkönnyíteni életünket egy bluetooth beacon? A parkolás és az autó azonosítása csupán egy a temérdek lehetőség közül. A személyes automatizálás új korszaka kezdődhetne, amelyben eszközeink intelligens módon reagálnak ránk és segítenék mindennapi életünket, például szólhatnának amennyiben tárcánkat otthon hagytunk volna.

Jelenleg az egyetlen járható út szakdolgozatom elkészítéséhez, amely ugyan nagyban korlátozza az alkalmazás nagyobb volumenekben való használatát és továbbfejlesztését is, hogy a megvásárolt beacon-öket a hozzájuk tartozó alkalmazással konfigurálom. Az alkalmazás későbbi továbbfejlesztése során feltétlen megoldást kell találnom a beacon-ök konfigurálásának kérdésére, az alkalmazásból, megkönnyítve ezáltal az alkalmazás használatát, hiszen nem elvárható egyetlen felhasználótól, se hogy két alkalmazásban is képes legyen beállítani egy megvásárolt bluetooth jeladót, úgy hogy azzal ne legyen semmi probléma.

Jelenleg csupán egyetlen kínai gyártó „kínál” akár külön alkalmazás nélküli kódból programozható beacon-t, bár az eszköz funkciói és a leírása egyaránt hiányos, és

amelynek működőképességében egyáltalán nem bízok, de lehet a későbbiekben kénytelen leszek majd használni.

A beacon-öket többféle üzemmódban lehet konfigurálni melyek befolyásolják az eszköz egy elemmel (vagy akkumulátorral) való üzemidejét. Ezen eszközök az előre beléjük táplált azonosítóikat előre meghatározott jelerősséggel és időközönként Bluetooth LE-t használva sugározzák. Ezen értékek testreszabásával lehetséges növelni vagy csökkenteni az ígért üzemidejét a készülékeknek.

Esetünkben bőven elegendő a beacon-t úgy beállítani, hogy azonosítóját a lehető legnagyobb időközönként (legritkábban) sugározza, illetve a teljesítményszintjét az autó fémszerkezetének elektromágneses sugárzásokra gyakorolt elnyelő hatása miatt a szükséges 5-10 méter helyett inkább 20 méterre konfigurálni, nehogy a túl gyenge jelerősség akadozó vétele az járművön belül is problémákat okozzon az alkalmazás számára. (Amennyiben az eszköz olyan zárt helyre kerülne a járművön belül, amelyet fémlapok határolnak bizonyos irányokból, például a kesztyűtartóba és a középkonzol fém elemei a túl gyenge erősségű jelet esetleg meg-meg szakítanák ezáltal parkolási esemény indítását kezdeményeznék az alkalmazáson belül)

Természetesen erre is fel kell készítenem az alkalmazást, és esetünkben a parkolási zónákban általános 5 perces türelmi időt kihasználva az akadozó jelre még csak egy időzítőt indítanék el, ráadásul az Awareness API és Core Motion segítségével le tudnám kérdezni a felhasználó mozgás állapotát, amely amennyiben nem változik „walking”/”ON_FOOT”-ra úgy ezen időzítőt leállítanám feltétezvén, hogy a beacon Bluetooth jele csupán időlegesen megszakadt. Fontos azonban tudni, hogy a telefon jelentése miszerint a felhasználójának a jelenlegi mozgás állapota a „következő”, időben késleltetve jelenik meg amikor a telefon a szenzoraiból nagy valószínűséggel meg tudja állapítani, hogy melyik mozgást végezzük. Ezeknek az konfidencia értékeit le tudjuk kérdezni mindkét platform által, azonban az egyszeri kiértékelésre a függvény a legvalószínűbb értéket adja vissza, amely rövid időtávon (a beacon sugárzási periódusa 1 mp leglassabb esetben) nem feltétlen egyezik meg a valósággal. Ezt figyelembe kell vennem az alkalmazás fejlesztése során.

Ezen fenti konfigurációk biztosítják a beacon és az alkalmazás számára a lehető leghosszabb és legmegbízhatóbb üzemidőt és a funkciók teljességét.

2.4 Fizetési megoldások

Magyarországon jelenleg minden online mobilfizetés, amely parkolásra irányul a Nemzeti Mobilfizetési Zrt szabályozása alá esik. A cégek külön szerződést kötnek a szolgáltatás igénybevételéhez. Jelenleg külön szerződéskötés meghaladja a lehetőségeimet a dolgozat készítésre, ezért egy egyszerűbb megoldásban gondolkozom, amely az SMS-es fizetés.

Irodalomkutatásom során arra jutottam, hogy egyik mobilplatform se nyújt olyan fizetési szolgáltatást, amely ne kívánná meg a felhasználó aktív részévételét a fizetési folyamat elindításában. Ez a felhasználó védelme érdekében érthető, hiszen így nem lehetséges annak tudta nélkül eltulajdonítani pénzbeli értékeit, azonban korlátozza az alkalmazás eredeti elképzelésem szerinti teljesen automatikus mivoltát, mert a felhasználó részéről aktív beavatkozást igényel.

2.4.1 Értesítés a fizetéshez

Ezen probléma kiküszöbölésének érdekében egy értesítéses rendszerben kívánom megvalósítani a fizetés, amennyiben az alkalmazás parkolási igényt észlel tájékoztatja értesítésben a felhasználót a szükséges lépés megtételére, és amely értesítésre rákattintva azonnal az alkalmazás „fizető” felületén találná magát az előre kitöltött adatokkal és csupán a küldés/fizetés gombra kelljen kattintania a parkolás megkezdéséhez.

Mivel jelenlegi világunkban elég sok értesítés érkezik a telefonunkra és a fizetés elmulasztása esetén büntetés jár, ezért amennyiben a felhasználó figyelmen kívül hagyná az első értesítést, a második már hangjelzés is követi, ehhez szintén engedélyeket kell kérni a felhasználótól.

2.5 Helyzetmeghatározás

Az alkalmazás működése számára kritikus fontosságú a helyzetmeghatározás. El kell tudni dönteni, hogy a jelenlegi tartózkodási helyzetünkön él-e aktív parkolási fizetési kötelezettség, illetve, hogy melyik parkolási zónában tartózkodunk éppen. A zónahatároktól függően más díjjakkal szembesülünk, illetve a zónahatárok összecserélése esetén, az esetleges parkolás indítás ellenére is büntetés fizetésre kerülhet sor. A technológia sajátossága, hogy pont városi környezetben a legpontatlanabb ezért különös tekintettel kell odafigyelni arra, hogy a kapott adatok nem biztosan fedik a valós tartózkodási helyet. Ezen funkciókat a beépített helyzetmeghatározó rendszerek (például GPS, GLONASS) és az elérhető NuGet package-ek segítségével tervezem megvalósítani.

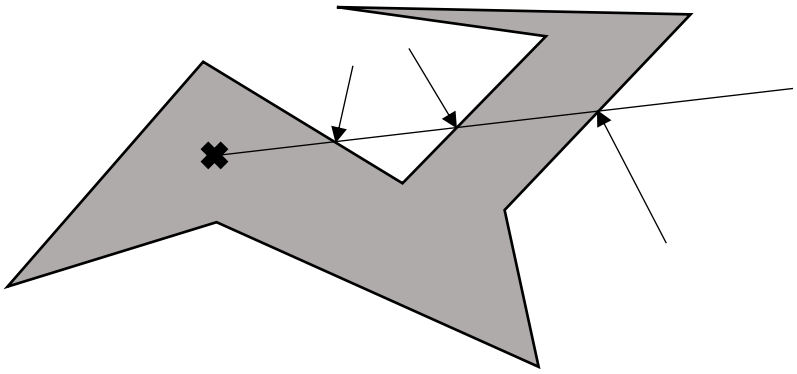
A Xamarin.Essentials NuGet package és az azon belül található GetLocationAsync és a GetLastKnownLocationAsync függvények segítségével a közös kódból tudom kezdeményezni a helyzetmeghatározást.

2.5.1 Zóna behatárolás, konvex és konkáv sokszögek

A Budapesti parkolási zónák többsége konkáv (az az egyes belső szögeik 180°-nál nagyobbak), mivel a terület, amiket határol a budapesti utcák elhelyezkedését követi. A zónákat, mint pontok által meghatározott egyenesek által határolt területeket tekintve, kell leellenőrizni, hogy éppen melyikben tartózkodunk.

Tanulmányaim során a szakirántárgyaim egyikén a Haladó algoritmusok című tárgy keretein belül sor került egy ehhez nagyon hasonló problémára az egyik beadandó feladat kapcsán, aminek a neve Smallest Boundary Problem volt. Ezen probléma megoldása során egy olyan megoldást kellett adni, ami a legkisebb korul határoló sokszög kerülete az előre megadott ponthalmazra úgy, hogy annak minden előre megadott pontot tartalmaznia kellett.

A probléma megoldása során minden iterációban el kellett dönteni, hogy az előre megadott összes pont benne van-e a következő lehetséges megoldás pontjai által határolt sokszög kerületein belül. Ezt úgy oldottam meg, hogy „ray casting” (sugár kibocsájtás) segítségével megszámoltam, hogy az adathalmaz bal széléről az előre megadott pontok hányszor keresztezik a sokszöget, és amennyiben páratlan kereszteződést számoltam meg úgy az előre megadott pont benne volt a sokszögben.



3. ábra, A sugárkibocsájtási módszer

Látható a fenti ábrán, hogy a sokszög belsejében elhelyezkedő pontból teljesen mindegy, hogy melyik oldalának irányába indítjuk a sugarat az, amennyiben benne helyezkedik a poligonban, mindig páratlan számú sokszor fogja keresztezni annak oldalait, amennyiben a sugarat a terület x vagy y irányú kiterjedésénél tovább vezetjük. Ezt megszámolva meg lehet állapítani, hogy a pontunk (tartózkodási helyzetünk) benne van-e a sokszögben.

Joggal merül fel a kérdés, hogy miért nem elegendő egyszerűen a zónahatárokat számolni, és amikor átlépünk egy határvonalat elkönyvelni, hogy akkor abban a zónában vagyunk mindaddig amíg át nem lépünk egy másik határt, amikor egy másik zónában vagy akár zónákon kívül lehetünk. Mivel mobileszközre szánt alkalmazásról van szó, és a zóna állandó számontartása és tudása nem járna az alkalmazás számára előnnyel, ellenben az

eszköz működési idejét negatívan befolyásolná ezért úgy döntöttem, hogy a parkolási esemény észlelésekor végzem el az ellenőrzést, kímélve ezzel is az üzemidőt.

Az üzemidő további kímélése érdekében a pozíció kiértékelésekor nem az összes zóna minden pontjaiból képzett zónákat vizsgálom, hanem egy leszűkített területen számolom a zónahatárok átlépésének számát. Ezt a leszűkítés azon alapul, hogy felesleges megvizsgálni minden olyan zónát amelyiknek a legdélebbi pontja nem éri el a tartózkodásihelyzet plusz mínusz a becsült pontatlanság által meghatározz zóna legészakibb pontját. És így tovább, mind a négy égtáj alapján el lehet végezni ezt a zóna szűrést, ezáltal is gyorsítva az alkalmazás futását és kímélve az akkumulátort.

3 SPECIFIKÁCIÓ

3.1 Az applikáció használt szoftverek

Fejlesztőkörnyezet: Visual Studio 2022

Android: 10-11-12 (teszteléshez)

Adatbázis: SQLite.NET[8]

3.2 Az alkalmazás funkciói

Parkolás indítás (Értesítésre)

Parkolás leállítása (Értesítésre)

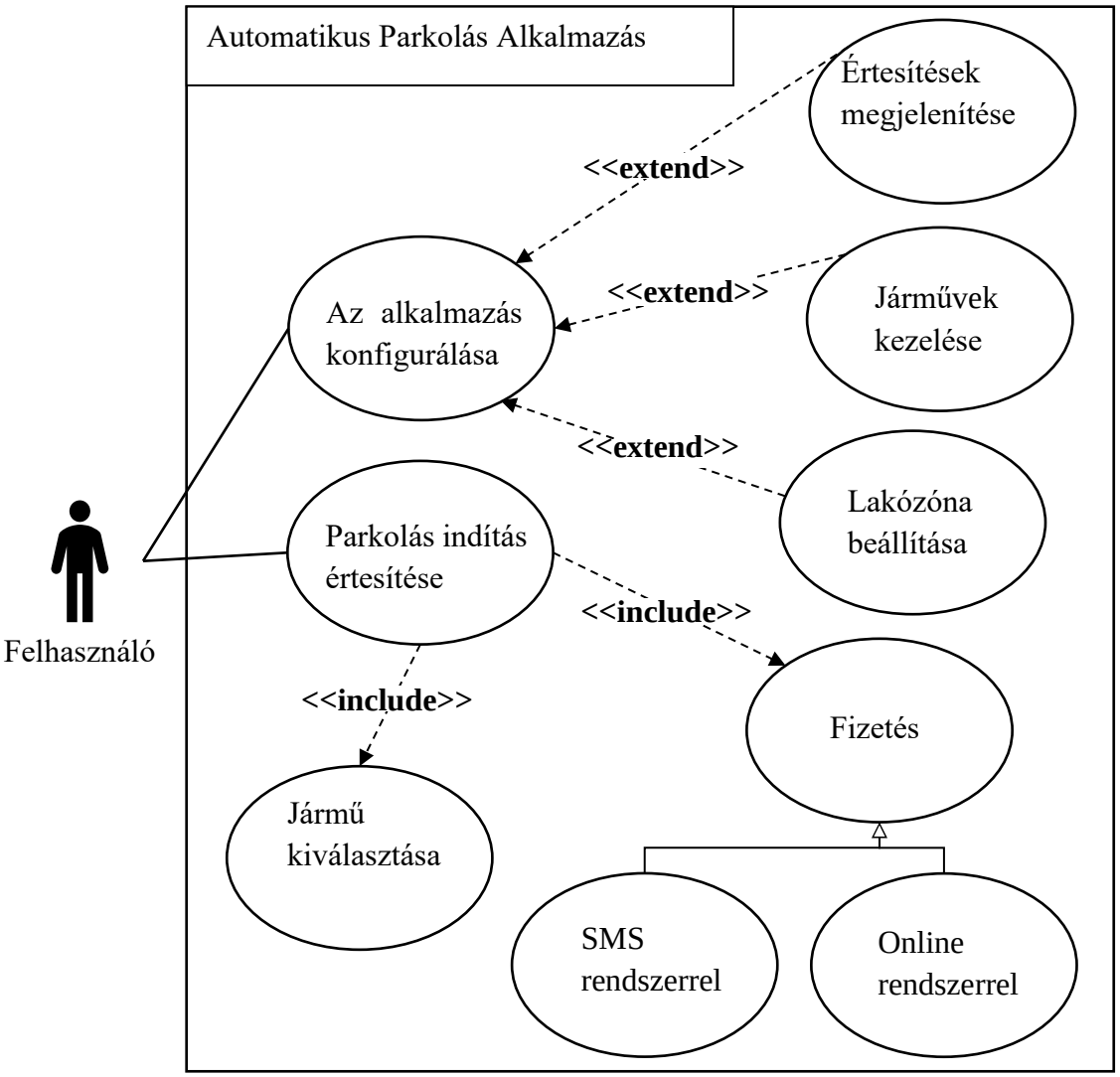
Járművek rögzítése, eltávolítsa

Könnyített fizetés indítás (SMS)

Parkoló Zónák megjelenítése

„Lakózóna”, mint ingyenes zóna beállítása

3.2.1 Az alkalmazás USE-CASE diagrammja



4. ábra, A USE CASE diagramm

4 TERVEZÉS

Az alábbi API-ok megvalósítása platformspecifikus kódot igényelnek. De a logika mögöttük hasonló.

4.1 Az Android Awareness API

A választott platformok sajátos megkötései miatt kénytelen vagyok az egyes platformokra külön platformspecifikus kódokat írni, hogy azok mindkét operációs rendszer alatt hasonlóképpen működjenek. Android alatt ezen szolgáltatáscsomagot az Awareness API nyújtja. [9]

A parkoláselindításhoz szükségünk lesz megállapítani a felhasználó mozgási módját és az abban beálló változásokat. Ehhez a legnagyobb segítséget az Awareness API-on belül a Snapshot API vagy a Fence API nyújtja. Ezek szolgáltatásait fogom használni mivel ez biztosítja számomra a figyelt megváltozott körülmények esetén a tájékoztatást az applikáció számára.

Az Awareness API-on belüli Snapshot API segítségével az aktuális állapotról tudunk tájékoztatást kérni.

Az Awareness API-on belül a Fence API segítségével olyan eseményfeliratkozásokat (Listener) tudunk létrehozni, amely összetett események együttes beállása/megváltozása esetén tájékoztatja az applikációt. Ezek többféle megkötések lehetnek:

- Idő alapú időzített
- Lokális, hely alapú, helyhez kötött - Geofence – virtuális kerítés
- Beacon BLE közelségalapú, amely egy eszköz által folyamatos időközönként sugárzott Bluetooth jel, amelybe való belépés és távozás esetén kap az alkalmazás a „tájékoztatást” – visszahívást.
- A DetectedActivityFence segítségével pedig a felhasználó mozgás módjának megváltozására „iratkozhatunk” fel. Ilyen lehet IN_VEHICLE vagy WALKING vagy ON_FOOT.[10]

Ezeket a megkötéseket és feltételeket kombinálni is lehetséges ÉS, VAGY, NEGÁCIÓ (NOT) kapcsolattal, kombinációjukkal olyan komplex események bekövetkezésére tudunk reagálni amelyek jelzik, a parkolási igény kezdetét és végét. Például parkolási időben eddig IN_VEHICLE volt az állapotunk, de megváltozott ON_FOOT-ra és a terület, ahol a változás történt parkolózóna. Ez két esemény fel és le iratkoztatásával lehet megvalósítani, mint komplementer események.

Az android rendszer ezeknek az API-oknak a használatát engedélykéréshez köti, amelyeknek megléte kritikus fontosságú az alkalmazás működéséhez, az ehhez tartozó

engedély az „android.permission.ACTIVITY_RECOGNITION” amelyet az AndroidManifest.xml-ben kell elhelyezni.

4.2 Apple Core Motion API

Hasonlóképpen az Apple iOS rendszerre is külön platformspecifikus kódot kell írnom a Xamarin megkötései miatt. Ehhez segítségül az Apple hasonló megoldását iOS 7.0-val bevezetett Core Motion Framework-ön belüli CMotionActivityManager-t fogom használni.

Az Apple biztonsági és tájékoztatási előírásai miatt az alkalmazás ahhoz, hogy használni tudja a hardverek által biztosított értékeket a Core Motion főoldalán leírtak szerint tájékoztatnia kell a felhasználót ezen értékek használatba vételéről az NSMotionUsageDescription nevű mező/leírás kitöltésével és engedélyt kell kérnie a felhasználótól. Ennek elmulasztása esetén a mozgáshoz kapcsolódó adatok puszta olvasási igényére az alkalmazás össze fog omlani.[11]

Irodalomkutatásom során beleásva magam az iOS engedélyek kezelésébe azt találtam, hogy vannak olyan esetek amikor a felhasználók véletlenül nem adják meg a jogosultságot az érzékelők használatára vagy időközben elveszik azt az alkalmazástól, ezért azt megelőzendő, hogy az alkalmazás kifagyjon az alkalmazás indításakor célszerű elvégezni egy ellenőrzést.[12] Az általam talált forrás egy másik framework engedélyeit ellenőrzi, a példában található engedélyt csupán le kell cserélni a CMAuthorizationStatus-ra.

A minket érdeklő része a Core Motion keretrendszernek a CMMotionActivity és a CMotionActivityManager amelyek az Android Awareness API-hoz hasonló funkcionalitást biztosítják iOS-en, így a keretrendszer biztosítja az alkalmazás számára, hogy ha elindítjuk a startActivityUpdates függvényt, akkor egy delegálton keresztül egy az arra feliratkoztatott függvény meghívódik a kiváltó eseményt pedig paraméterként megkapjuk és ennek megfelelően cselekszünk tovább.

Ezek a CoreMotionActivity-k tulajdonképpen boolean-ok az az igaz-hamis igazásgértékek lehetnek például: stationary (egyhelyben áll), walking (gyaloglás), automotive (járműben van). Ezek adatok értelmezésnél fontos, hogy egyszerre akár többnek is lehet igaz az értéke.

4.2.1 Apple Core Location keretrendszer a virtuális kerítésekhez és iBeacon detektáláshoz

Az Android Awareness-ben tapasztalt virtuális kerítés funkciót (geofencing) az Apple Core Location tartalmazza, erre is ugyan úgy érvényesek a jogosultság szerzési igények. Ezen felül ez a keretrendszer tartalmazza az iBeacon-ök detektálásának funkcióit is.[13]

Az alkalmazás használatához feltétlen szükséges, hogy az alkalmazás akkor is kapjon tájékoztatást a a virtuális kerítés, illetve az iBeacon hatósugarában való tartózkodásról, ha az éppen a háttérben fut vagy egyáltalán nem is fut. Biztonsági okokból az Apple reagálva bizonyos felhasználók félelmeire azok csökkentése érdekében bevezette a többszintű engedélyezési rendszer, amellyel több lépésen keresztül kell a felhasználónak jóváhagynia az alkalmazás számára a GPS illetve iBeacon használatot.[14]A locationManager.requestAlwaysAuthorization() használatától függetlenül a rendszer először lehetőséget sem ad a felhasználónak, hogy biztosítsa a szükséges felhatalmazást az alkalmazás számára, Ezt többlépcsőben teheti csak meg ezáltal csökkentve a felhasználó élményt.

A geofencing funkciót a CLLocationManagerDelegate-re való függvények feliratkoztatásával lehet megvalósítani, amely minden körülmények között értesíti a háttérben futó alkalmazást, illetve az adott esetben kikapcsolt alkalmazást el is indítja a háttérben, hogy funkcióját elláthassa.

Az iBeacon-ök érzékelését a CLBeaconRegion valósítja meg amellyel egy a mi általunk előre az iBeacon eszközbe programozott UUID, valamint major és minor értékekkel rendelkező iBeacon-öket tudunk kerestetni az alkalmazással. Ez az osztály is képes felébreszteni és háttérbe futtatni az alkalmazást.

4.3 Események és kezelésük

A funkciók ellátása a komplex események regisztrálása az operációs rendszerekben egy gyors és kényelmes megoldás. Az összetett feltételek bekövetkezésekor kiváltódó eseményre reagálva megvalósíthatóak az alkalmazás funkció, azonban ezen eseményeket semmi nem akadályozza meg, hogy folyamatosan kiváltódva mind gátolják az alkalmazás futását, mind feleslegesen erőforrásokat emésszenek fel az eszközből. Ezért ezen eseményeket folyamatosan módosítani kell, felregisztrálni, leregisztrálni. Két fő eseményt tervezek a rendszerbe és mindegyik komplex feltételrendszer együttes megvalósulása esetén fog kiváltódni. Ezek komplementer eseményeknek számítanak, az az egy időpillanatban egy fog lenni feliratkoztatva az operációs rendszerek eseménytárában. Az egyik esemény kiváltódása előidézi saját maga törlését az eseménytárból és a másik esemény létrehozását és regisztrálását.

A main() függvényben alapból feliratkoztatott kezdő esemény a drivingEvent().

Az energiaforrásokkal való spórolás miatt az egyes feltételek sorrendjét úgy határozom meg, hogy a legszámításigényesebb feltételekre (helyzetmeghatározásra, geofencingre) csak akkor kerüljön sor, ha és amennyiben az összes többi feltétel adott. Mindkét rendszerben lehetséges komplex feltételű eseményeket regisztrálni, az egyes események logikai feltételei azonban a rendszerfüggetlenek és ezeket a feltételek a következők eseményekre lebontva a kiértékelési sorrendjében.

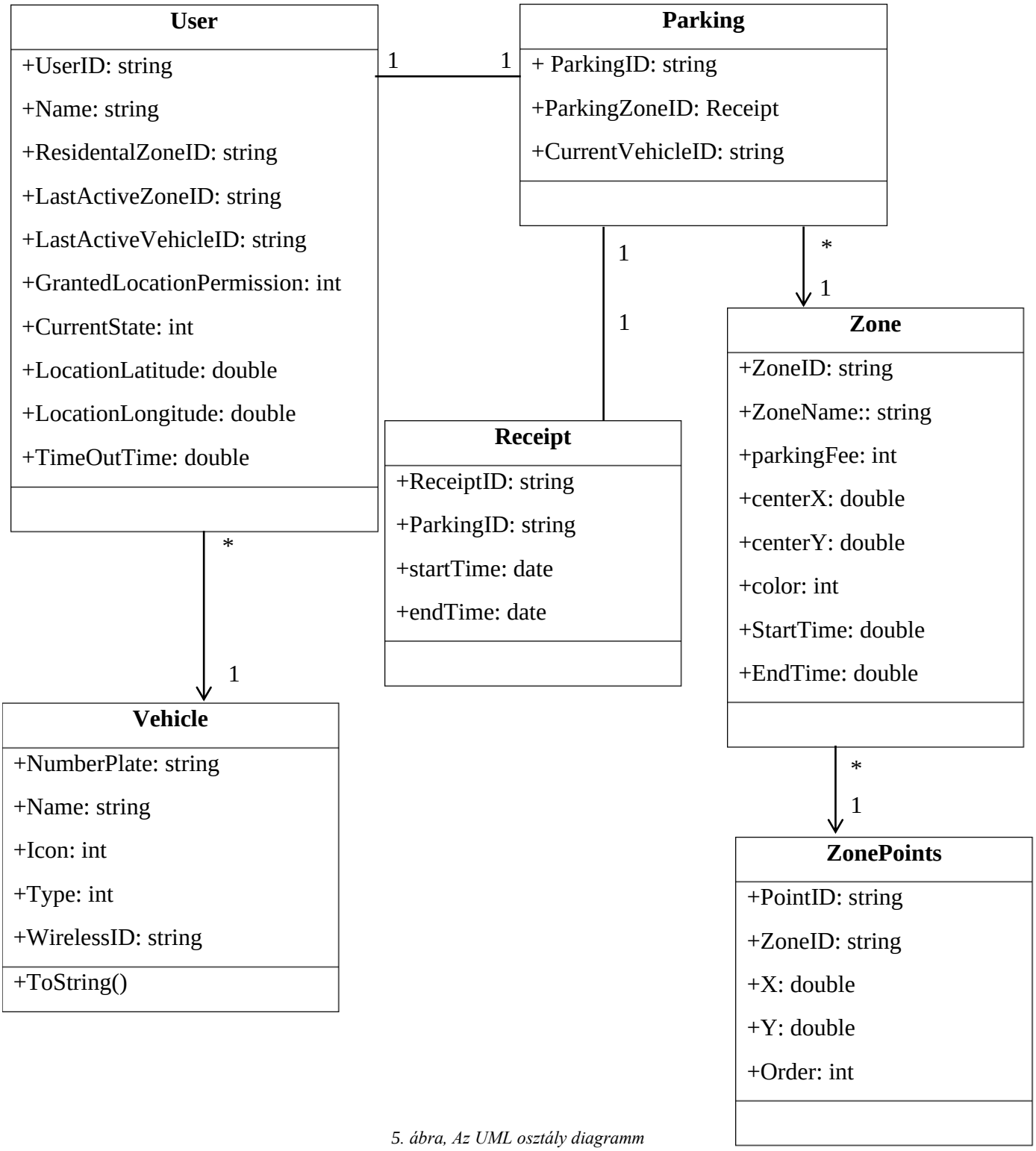
Walking event:

- Időfeltétel, parkolási időszak
- Vezetéknélküli azonosító jelének hiánya (amennyiben létezik)
- Mozgás állapot: gyaloglás

Driving event:

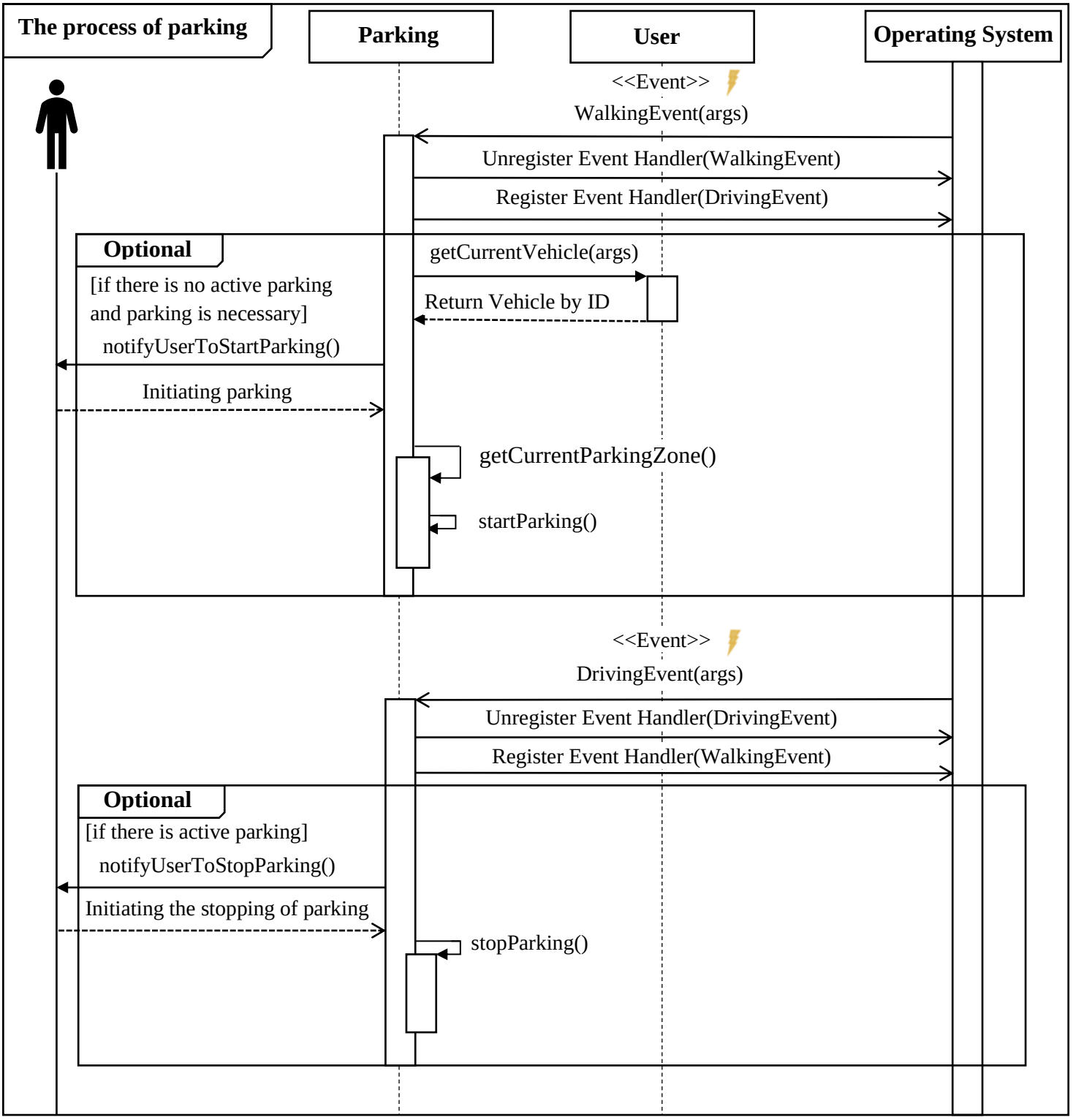
- Időfeltétel, parkolási időszak
- Vezetéknélküli azonosító aktív jele (amennyiben hozzákapcsoltunk ilyet a járműhöz)
- Mozgás állapot: járművön
- Geofence: A legutolsó jármű legutolsó ismert helyének virtuális határain belül tartózkodunk

4.4 Az alkalmazás UML osztály diagrammja



5. ábra, Az UML osztály diagramm

4.5 Az alkalmazás UML szekvencia diagrammja



6. ábra, Az UML szekvencia diagramm

5 A MEGVALÓSÍTÁS LEÍRÁSA

A megvalósítás során két olyan esetet tudok kiemelni, amely hátráltatta a megvalósítást.

5.1 Hátráltató tényezők

5.1.1 Az SQLite

A megvalósítás során számos akadályba ütköztem melyek hátráltatták a funkciók megvalósítását, illetve végül megakadályoztak abban, hogy bizonyos funkciókra megfelelő időt fordítsak. A dokumentációk és az információk teljes vagy részleges hiánya miatt sok mindent csak nagy mennyiségű manuális, illetve valós körülmények között elvégzett teszteléssel tudtam megvalósítani, illetve rájönni a hogyanjára.

Az egész megvalósítás során hátráltatott az SQLite könnyűsúlyú adatbáziskezelő rendszere, amelyből hiányoznak olyan SQL-ben meglévő alapvető funkciók, mint a DROP DATABASE. Emiatt az esetleges időközbeni (elég sűrű) adatmodell változások során adatbázishibák tömkelegével találkoztam, amelyekre a megoldást kezdetben az Android emulátor-ban lévő készülék gyári visszaállítása jelentette az egyetlen gyors megoldást és amely későbbiekben a választott Awareness API működéséből kifolyólag (amely csak is kizárólag valós készülékeken hajlandó adatot szolgáltatni valós körülmények között és amelynek a dokumentációjában erről egy szó nem esett) a megvalósítás késői szakaszában bekövetkezett modell változtatást rémálommá tette, ugyanis a számomra elérhető készülékek egyikén se állt módomban gyári visszaállítást eszközölni. Ez az Android Studio bevetéséhez vezetett melyet a gyárilag elérhetetlen mappákban lévő SQLite adatbázis törlésére használtam.

Az SQLite aszinkron mivolta miatt a megvalósítás során sok holtpont („deadlock”) keletkezett amikor az alkalmazás indításakor a szinkron felületeknek el kellett érniük az aszinkron adatbázist. Ezeknek a megoldására külön Task-okon indítottam el a szinkron felületekről az adatlekérést.

5.1.2 Az elérhető dokumentációk hiánya

A választott nyelv és fejlesztői környezet a dolgozatban eddig említett hátrányaihoz a megvalósítás után hozzátenném, hogy bár az elérhető Androidfunkciók relatív naprakészek voltak, ezeknek a használatához nem tartalmazott, csak rendkívül régről származó elavult dokumentáció, amelyet nem lehetett felhasználni semmire.

Az elérhető NuGet package-ekben használt Awareness API kezelését megvalósító függvények pontos használatát a szakdolgozat megvalósításának idejében egyáltalán semmilyen ahhoz kapcsolódó dokumentációban nem találtam meg. Ezen függvényeket és használatukat egyetlen az interneten fellelhető forrásban lévő, de abban is egy elavult

verziónak a kódszintű megvalósításának elemzésével tudtam egyáltalán megállapítani, hogy hogyan hívják azt a függvényt, aminek a visszatérési értéke „IFenceUpdateRequest” típusú és amit így elfogad az UpdateFences() függvény, mert ezt az információt nem tartalmazza a függvény kódból elérhető metadata-ja, de ezt az interfészt sehol máshol az egész interneten fellelni nem lehet.[15] Az elavult fellelhető információ miatt sajnos magamtól és az IntelliSense-ből elérhető függvény és annak elvárt bemeneti paramétereinek kombinálásból kellett kitatálnom, hogyan használjam azt. Természtesen „Trial and Error” módszerrel és ezeknek a teszteket elvégezését tovább súlyosbította az Awareness API-nak az a tulajdonsága, hogy csak valós körülmények közötti szolgáltat adatokat, ami azt eredményezte, hogy a valós tesztelés és a függvények megvalósítása ezentúl kétemberes feladattá vált, amely nagyban megnehezítette a haladás tempóját.

Az Awareness API dokumentációja Xamarin specifikusan hiányos, ez lehetséges, hogy csak az általam választott platform egyik nemvárt hátránya. de ezek a dokumentációk elvárnak ismereteket az Android Intent-jeivel és PendingIntent-jeivel kapcsolatosan.

Az Awareness API egyik dokumentációjában sem említik, hogy szükség lenne, a visszahívás („Call Back”) kezeléshez, egy bizonyos „RegisterReceiver()” függvény megfelelő paraméterekkel való meghívására. Ez számomra az egyetlen elérhető internetes forrás mélyreható kódelemzésével derült ki. És mivel az egy elavult formulának a megvalósítását tartalmazta csak reménykedtem, hogy ebben a verzióban is működni fog.

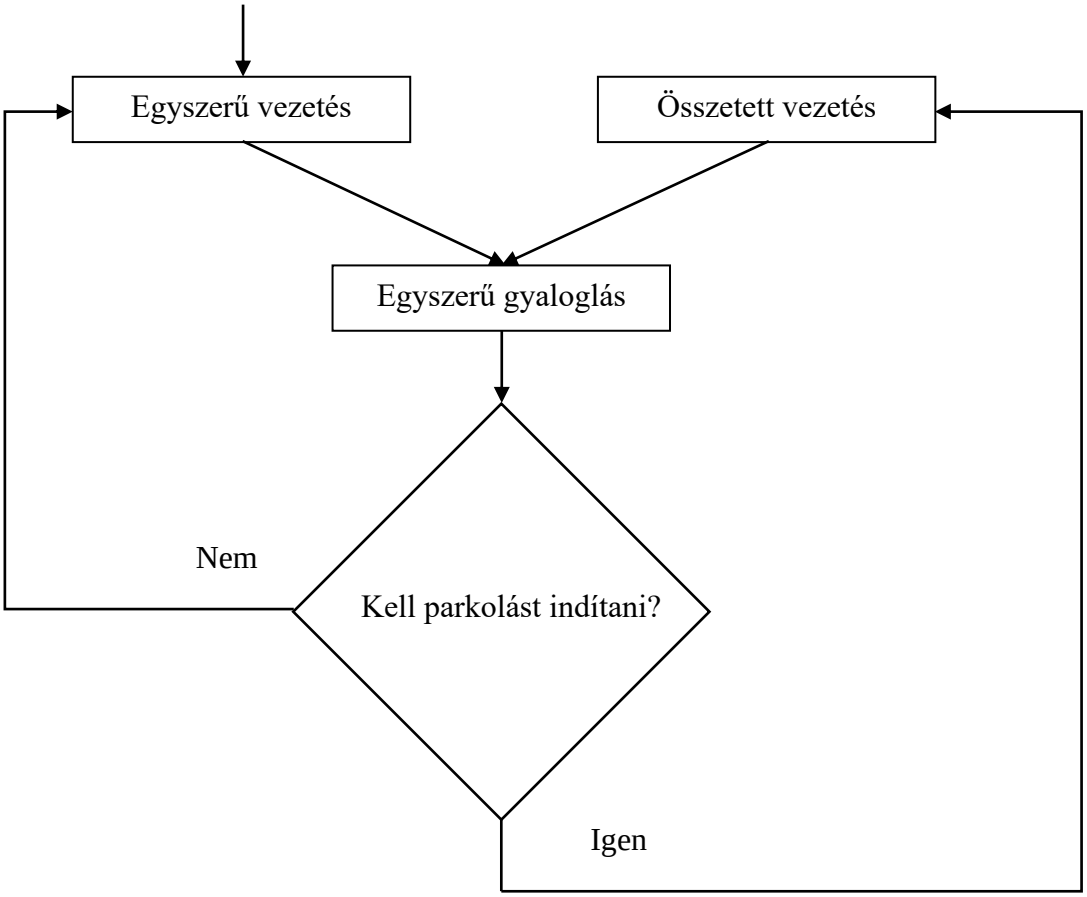
Az Awareness API Xamarin oldalról való megvalósítása ezentúl is számtalan váratlan akadályba ütközött, nem mindegy hogy milyen régen lett regisztrálva az előre meghatározott esemény fogadására a „RegisterReceiver()” függvénnyel a megfelelő BroadcastReceiver osztály, mert bizonyos idő után az Android rendszer az eseménykezelő osztály regisztrálását „inaktiválja” így nem lehetett tudni, hogy vajon azért nem jött meg az értesítés, mert nem működik az API, vagy mert régen regisztráltam a rendszerben, vagy elrontottam a meghívást, egyáltalán be van-e regisztrálva. Ezeknek az lehetőségeknek a kiszűrését természetesen az Awareness API miatt valós körülményeket között kellett végezni.

5.2 Az események megvalósítása

Az dolgozatban fentebb említett két esemény a valósítás során kiderült, hogy elégtelennek bizonyultak a feladat ellátására, ezért be kellett vezetni egy harmadik eseményt is. Az eredeti tervek szerint két esemény látta volna el a parkolás detektálási funkciókat egy gyaloglás „WalkingEvent” és egy vezetés „DrivingEvent”. Viszont a megvalósítás során rájöttem, hogy a vezetés eseményét szét kell szedni, egy egyszerű és egy komplex vezetési esemény bekövetkezésére. Ezeket a vezetési eseményeket a gyaloglás esemény indítja el attól függően, hogy épp parkolási zónában tartózkodunk-e, illetve parkolási idő van-e. Amennyiben igen úgy az összetett parkolási eseményt hozza létre és állítja be annak kezelését. Amennyiben olyan helyen vagy időben álltunk meg ahol nincs fizetés

akkor az egyszerű vezetési eseményt állítja be következően várt eseménynek és nem értesíti a felhasználót.

Az alábbi ábra az elvárt eseményeket mutatja, nem közli, hogy maguk az események milyen feladatokat hajtanak végre.



7. ábra, Az események sorrendje, megvalósítása

Például az összetett vezetés esemény bekövetkezése esetén, leállíttatja a parkolást, illetve az egyszerű gyaloglás esemény „elindítja” valamelyik vezetés eseményt, az összetettet, ha szükséges a parkolás és jelzi a felhasználónak, hogy indítsa el a parkolást.

5.3 A zóna behatárolás megvalósítása

Sajnos az előzőekben általam megvalósított sugár kibocsájtási megoldást nem tudtam egy az egyben alkalmazni ezért az idő szűke miatt egy az interneten talált hasonló megoldást használtam, amely működőképesnek bizonyult.[16]

5.4 Activity Recognition API (Client)

A megvalósítás és a tesztelés párhuzamosan folyt, ennek megfelelően időben értesültem, az Awareness API lassúságáról, új, pontosabb megoldások felé néztem és ekkor jött szembe velem Activity Recognition Client és annak két dolgozatom szempontjából fontos API-ja a Transition API és Sampling API.

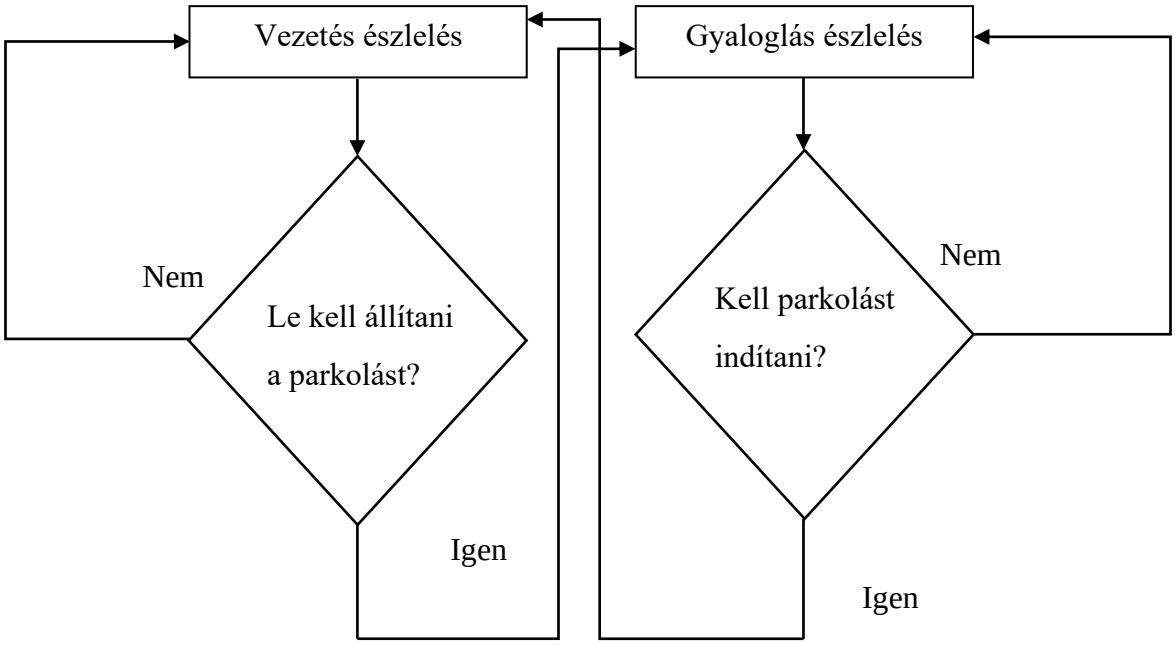
A Transition API hasonlóan az Awareness egyik funkciójához képes arra, hogy előre beállított mozgásállapotok közötti átmentet detektáljon. Beállítható, a cselekvés maga, illetve a tájékoztatás időpontja, hogy a cselekvés kezdetén értesítsen-e vagy a végén. A Transition API-al, mint kiderült, ugyan az a probléma, mint az Awareness-el, kikapcsolt kijelző mellett lassan küldi az értesítéseket, így a továbbiakban a Sampling API-t és megvalósításának leírását szeretném bemutatni.[17]

A Sampling API a dokumentáció szerint képes arra, hogy előre meghatározott időközönként tájékoztatást adjon a mozgás állapotról a „requestActivityUpdates (long detectionIntervalMillis, PendingIntent callbackIntent)” függvény segítségével, a long érték meghatározásával. Az elérhető dokumentációban leírják, hogy habár ez a long érték megadható, ettől az értéktől való eltérések előfordulhatnak, mindkét irányba, előfordulhat, hogy a beállított értéktől gyakrabban vagy ritkábban kapunk frissítéseket.

Viszont ez az API olyan összetett feladatok ellátására képtelen, mint amire az Awareness API-t használom, így a pontosabb működés reményében átdolgoztam az alkalmazást.

A Sampling API csupán a telefon legvalószínűbb mozgás állapotairól küld tájékoztatást a beállított intervallumonként, így az Awareness eseményeit nem lehet hozzá használni mivel csak arról kapunk tájékoztatást, hogy a felhasználó mit csinál, gyalogol, vezet. A geofencing-et és az idő ellenőrzést és a többi funkciót így mind-mind máshogy kellett megvalósítani.

Erre ezzel az API-al csupán egyetlen BroadcastReceiver-t használtam, amit kétkülönböző IntentFilterrel „hívatok” meg, és soha nincs egyszerre a kettő, így spórolva az erőforrásokkal.



8. ábra, Az Activity Recognition Sampling API megvalósításának logikai ábrája

Az alkalmazás ezen átírt verziójában már valóban a 4.5-ös pontban lévő eseményeke alkalmazásával valósítom meg a megoldást.

5.5 Egyéb tényezők

Fontosnak találtam megemlíteni, hogy a megvalósítás során próbáltam az időben hátrányos adatbázis műveleteket többnyire aszinkron módon megvalósítani, emiatt, az implementálás kezdeti fázisaiban relatív sok „deadlock”-ra az az holtpontra futottam. Ezeket külön szálak indításával orvosoltam, amelyek futásának végén térek vissza az eredménnyel

6 TESZTELÉS

Az alkalmazás tesztelése sajnos nem volt lehetséges automatizálta, mert a használt API-ok nem reagálnak csak valós körülményekre. A kód manuális tesztelésének nagy része arra irányult, hogy a dokumentáció hiánya miatt próbáltam rájönni a megfelelő „használatra”. Az általam írt függvények pedig olyan egyszerű „megfogalmazások” melyek tesztelését nem tartottam indokoltnak, unit tesztek keretében megvalósítani, mivel túlságosan egyszerűnek hatottak volna, a logika maga nem volt komplikált, ezért nem növelte volna a kód megbízhatóságát, minőségét.

A tesztelés nagyrésze abban merült ki, hogy próbáltam rájönni a tőlem független API-ok belső működésnek lelki világára, és rábírni őket a működésre.

6.1 Az Awareness tesztelésének leírása

Az Awareness API, mint kiderült nem reagál, csak a telefon beépített gyorsulásérzékelőjének jeleire, és más módszert nem találtam, amivel befolyásolni lehetett volna azt, hogy a telefon jelezzen a feliratkoztatott BroadcastReceiver-eknek, hogy esemény történt. Bár megpróbáltam az ADB segítségével a saját Intent-emre kiváltani eseményt – sikertelenül.

Kipróbáltam GPS szimulátor alkalmazását, amely mozgás állapotot is tudott volna elvileg szimulálni, de sajnos nem se járt sikerrel az Awareness API átverésében.

Így a tesztelés teljesen manuálisan zajlott, vezetési esemény kiváltására vezetni kellett, gyaloglás eseményhez pedig gyalogolni, miközben a debuggolás miatt kábeles összeköttetés kellett a laptop és a telefon között. Ezért a tesztelés rendkívül lassan zajlott. Ahogy korábban is említettem a fejlesztés és a tesztelés nagyjából együtt zajlott, muszáj voltam egyszerre csinálni, mivel a funkciók megvalósítása során sokat kellett próbálkozni milyen módon tudnának működni. Sokszor a „terepen” kellett fejleszteteni.

A tesztelés során kiderült, hogy az Awareness API nem képes feladatát 100%-osan ellátni, ugyanis, mint utólag kiderült azt akkumulátor kímélően valósították meg, ezért bár fel van iratkozva esemény a megfelelő paraméterekkel, a mozgás állapotban való változást detektálni, értesíteni az alkalmazást kikapcsolt kijelző esetén csupán nagy késéssel átlagosan 5-10 perccel később képes, ha egyáltalán érkezik értesítés. Azonban amint a képernyő felvillan és aktivizálódik a telefon, az értesítés azonnal megjön az állapotváltozásról.

A tesztelés során több valódi telefonra is feltöltöttem az alkalmazást, eközben derült ki, hogy néhány telefonon máshogy jelenik az alkalmazás, mint az emulátorban, alap színbeállítások miatt nem látszódtak a szövegek.

6.2 Az Activity Recognition Sampling tesztelésének leírása

A tesztelés során arra az eredményre jutottam, amennyiben a kijelző aktív az alkalmazás ellátja funkcióját és autóból kiszállva 10-15 méteren belül érkezik az értesítés és járműbe ülve nagyjából 1 perc vezetés után szól, hogy le kéne állítani a parkolást.

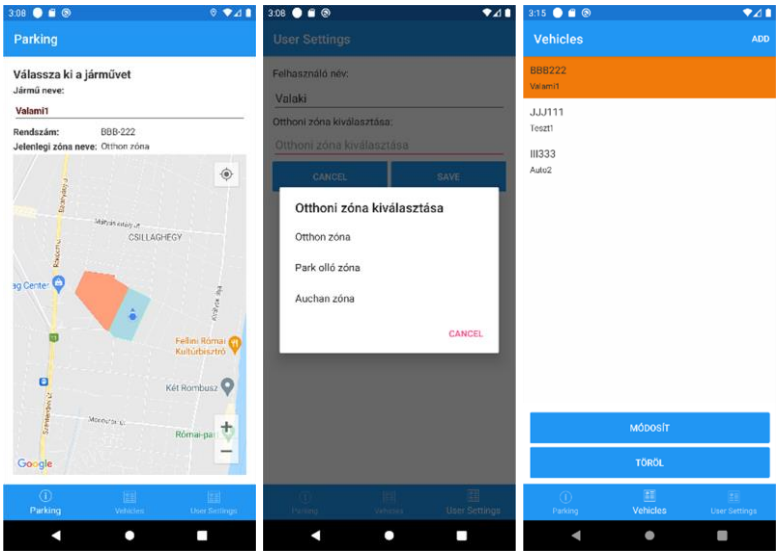
Kikapcsolt kijelző esetén az előre beállított 15 mp-es értesítési időközökből 40 mp-től 80 mp-ig terjedő időtartamra bővül a valós időköz. Sajnos az eszköz pontatlanságából kifolyólag az észlelt tevékenység nem feltétlen azonos a valós cselekvéssel. Ezt úgy értem, hogy a Activity Recognition Sampling API által, Intent formájában a BroadcastReceiver-nek továbbított észlelt állapotok többfélék lehetnek, melyek közül kettőt használok, az ON_FOOT és az IN_VEHICLE állapotokat.

Ezeket leszámítva még további 6 féle állapotot tud közölni velünk az API, és olyan a tesztelés több hete alatt egyszer se fordult elő, hogy miközben gyalogoltam, IN_VEHICLE vagy vezetés közben ON_FOOT jelentés érkezett volna. Csupán olyan, hogy miközben autóban voltam, UNKNOWN vagy STILL vagy TILTING jelentés érkezett, amikkel a BroadcastReveicer-ben nem is foglalkozik, így hibás működéshez, a nem pontos állapot jelentés nem vezet.

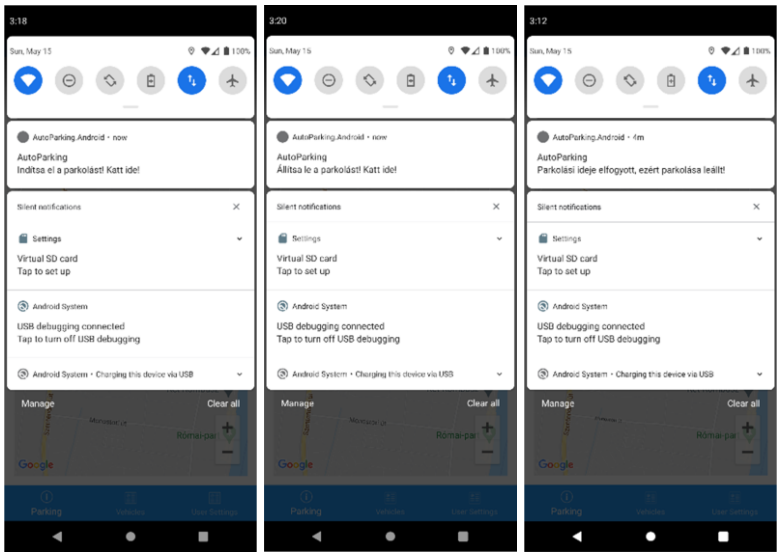
A cselekvés pontos észleléséhez lehetséges, hogy több ciklust is végig kell várni, ami így többször 1-2 percet vesz igénybe, de ez az elindítás során bőven az 5 perces türelmi időn belülre esik, illetve leállítás során pusztán pár percnyi túlfizetést.

7 EREDMÉNYEK BEMUTATÁSA

Az alkalmazás eredeti funkcióját ellátja, képes a felhasználó mozgás állapotából kikövetkeztetni a parkolás indításának szükségességét. A kiválasztott járműre parkolási esemény észlelése esetén az alkalmazás tudatos használata nélkül értesítésben értesíti a felhasználót, hogy indítsa el a parkolást melyet a felhasználó az értesítésre kattintva tud elindítani SMS-ben, amely előre kitöltötten várja és csak a küldés gombra kell kattintania. Új jármű rögzítésére képes. Az előre megadott lakó zónában nem küld értesítést parkolásindítása.



2. Kép Az alkalmazás funkcióinak összefoglalása



1. Kép Az értesítések, amelyek tájékoztatják a felhasználót a teendőkről

A leállításra is értesítés érkezik melyre rákattintva szintén kitöltött SMS várja a felhasználtót. Így az alkalmazást tulajdonképpen csak egyszer kell elindítani, rögzíteni a jármű adatait, onnantól képes magától működni, és küldeni az értesítéseket az előre kitöltött SMS-ekkel.

Az alkalmazás képes visszaállítani önmagát egy esetleges szándékos leállítás után is, a megfelelő „pozícióba” így, ha eddig parkoltunk akkor továbbra is a parkolás végét fogja vizsgálni.

Sajnos az SMS-beni fizetés nem éppen a legfelhasználó barátabb pénz szempontjából, ugyanis drága. Ellenben az alkalmazás funkcióját más módon nem tudtam megvalósítani ugyanis szerződéskötés kellene a Nemzeti Mobilfizetési Zrt.-vel, amelyek meghaladják lehetőségeimet. Így az alkalmazásom számára az SMS fizetés az egyetlen megmaradt opció.

7.1 A mérési eredmények táblázatos formában

Az parkolás indítás érzékelés gyaloglás és a további feltételek meglétét jelenti.

A parkolás leállítás, a vezetés megkezdését és a terület elhagyását jelenti.

Awareness API	Érzékelés elvárt ideje	Mért idő (Debug)	Mért idő (Önmagában)
Parkolás indítás érzékelése (Aktív kijelző)	15 mp	20-40 mp	10-30 mp
Parkolás leállítás érzékelése (Aktív kijelző)	15 mp	40-60 mp	30-60 mp
Parkolás indítás érzékelése (Kikapcsolt kijelző)	1 perc	10+ perc	nem érkezik értesítés legalább 10 percig
Parkolás leállítás érzékeléseű (Kikapcsolt kijelző)	1 perc	10+ perc	nem érkezik értesítés legalább 10 percig

1. Táblázat, Az Awareness elvárt és mért eredményei

Látható, hogy miért volt szükség az Awareness lecserélésére, ugyanis az kikapcsolt kijelző mellett gyakorlatilag használhatatlan eredményeket produkál, ezért valósítottam meg az ARS API-val az alkalmazás funkcióit. amely láthatóan konzisztensebb értékeket hoz és képes kikapcsolt kijelző mellett is működni.

Sampling API	Érzékelés elvárt ideje	Mért idő (Debug)	Mért idő (Önmagában)
Parkolás indítás érzékelése (Aktív kijelző)	15 mp	15-30 mp	15-30 mp
Parkolás leállítás érzékelése (Aktív kijelző)	15 mp	15-30 mp	15-30 mp
Parkolás indítás érzékelése (Kikapcsolt kijelző)	1 perc	1-2perc	1 perc -2 perc
Parkolás leállítás érzékelése (Kikapcsolt kijelző)	1 perc	40 mp-1 perc 30 mp	1 perc -2 perc

2. Táblázat, Az Activity Recognition Sampling elvárt és mért eredményei

8 TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

Az egyik terület, amiben az alkalmazást tovább lehetne fejleszteni, az a térkép valódi adatokkal való feltöltése, bár ezeknek az adatoknak a hozzáférhetősége kétséges, az adatmodellt mindenképpen át kéne alakítani a valós adatok kezeléséhez.

El tudnék képzelni olyan továbbfejlesztési lehetőséget az alkalmazás számára a későbbiekben, hogy automatikus fizetés indítást lehetne kezdeményezni, akár automatikus SMS akár későbbiekben interneten megvalósítandó fizetéssel.

A zóna választás manuális pontosítását választással is ki lehetne egészíteni a kritikus zónahatárok és a GPS esetleges pontatlansága miatt, főleg a belvárosban tapasztalható olykor egészen apró zónák miatt.

A megvalósítás során sajnos csak az Android-ra való fejlesztésre voltak meg a forrásaim, mivel az iOS-re való fejlesztéshez Mac rendszerű számítógép szükséges, amivel sajnos nem rendelkezem. A továbbfejlesztés során biztosan kifejlesztésre kellene kerülnie az iOS verziónak is.

Továbbfejlesztési lehetőség továbbá a fedélzeti Bluetooth és beacon-ök az alkalmazásba való beépítése is, melyeket a fejlesztés során nem tudtam beépíteni az alkalmazásba, pedig nagymértékben csökkenthették volna az alkalmazás reakció idejét és növelhették volna pontosságát.

Ez a parkolási és gyaloglási esemény gyorsításán kívül arra is jó lenne, hogy automatikusan ki tudná választani a parkolás indítás járművét, így azzal se kellene foglalkozni. A Bluetooth arra a problémára is megoldást jelentene, hogy ha a felhasználó a saját járművével leparkol, de másik eszközzel indul tovább, úgy nem keletkezne felesleges értesítés.

9 ÖSSZEFOGLALÁS

Dolgozatomban arra a problémára kerestem megoldást, hogy manapság a parkolás indítás és leállítás manuális feladat, melyek elmulasztása pénzbüntetést vagy túlfizetést von maga után. Erre egy parkoló alkalmazás megvalósításával próbáltam megoldást keresni, amely problémát több oldalról közelítettem meg. Először feltérképeztem a hardveres majd a szoftveres parkolás érzékelés/indítási eszközök tulajdonságait, megbízhatóságát, biztonságát, költségvonzatait. A jelenlegi tudásom alapján döntve a választásom a fejlesztői környezet esetén a Visual Stuido-ra, míg nyelvként a C# Xamarin-ra esett. Ennek a választásnak előnyei és hátrányai is voltak.

A szoftveres megoldást részesítettem előnyben, mert funkciót tekintve kevéssel marad el megbízhatóságban egy hardveres CAN BUS érzékelőtől, de biztonság tekintetében jóval biztonságosabb. A szoftveres megoldás során az Androidra alkalmazásra koncentráltam, mert ezzel volt korábbról tapasztalatom és mivel nem rendelkezek iOS-re való fejlesztéshez megfelelő eszközparkkal.

Az Android alkalmazás megvalósítás során láttam a Xamarin hátrányait, amelyek leküzdése némely esetben igencsak nehéznek és időrablónak bizonyultak. Nem volt elérhető dokumentáció. Az Androidon elérhető Awareness API amelyet a mozgás állapot detektálására és komplex események összekapcsolására használtam megakadályozták az automatikus tesztelést ez lassította a fejleszt is mivel a dokumentáció hiánya miatt nagyon sok mindent csak a „terepen” tudtam tesztelni/fejleszteni, manuális teszteléssel, lassabb hordozható számítógéppel, kábellel összekapcsolva miközben sétálni kellett egy valódi Androidos készülékkel.

Ez, mint később kiderült, kikapcsolt kijelző mellett nem működőképes, így az Activity Recognition Sampling API-t használtam a továbbiakban, amely egy jobb választásnak minősült az alkalmazás funkciójának ellátásának tekintetében.

Így az átdolgozott alkalmazás jelenlegi állapotában működőképes 30 mp-en belül értesíti a felhasználót amennyiben az használja épp a telefonját, illetve 1-2 perc késessel, amennyiben a kijelző kikapcsolt állapotban van. A továbbfejlesztési lehetőségek megvalósításával olyan alkalmazás születne, amely képes kiváltani a jelenleg megszokott parkolási rutinunkat és olyan megoldással helyettesíti azt, amely nem igényelne a felhasználó részéről aktív beavatkozást.

10 CONCLUSION

In my thesis I have tried to find a solution to the manual start and stop of parking in Budapest because the existing solutions demand a conscious decision to start or stop the parking and in case of someone forgot either of the above it would result fines or overpaying. The solution that I have imagined is an application that would notify the user in case of them leave their vehicles and start walking in a parking zone during payment period. First, I have searched solutions to „sense" when to start/stop parking in the hardware level connecting to the car's CAN BUS system to determine its closed/opened state. There were security issues with this, so I have focused all my attention to a software-based solution. As my current knowledge is focused on C# it was obvious that I will choose Xamarin and Visual Studio for development. This fact has its benefits as its drawbacks.

Going with a software-based solution I have limited myself for a less accurate and slower sensing of the necessity of starting/stopping the parking. But for security reasons which a hardware-based CAN BUS reader could cause this was the only solution remaining.

As developing to iOS is requiring a Mac and fees the only remaining possibility were to develop only to Android. Combined with Xamarin, Android application development can be timeous because the availability of current APIs and knowledge/documentation of how to use them is limited at best. On Android I have used the Awareness API to determine the device movement state (walking, driving) and to combine them into complex events with time and location. The lack of documentation was frustrating, and it was time consuming to solve the problems that have emerged as development goes forward because Awareness is only working on physical devices during real circumstances like if we expect a driving event, we had to drive. It meant that testing would be a manual process in the field with a laptop with cables dangling while walking and driving and occasional debug sessions was interrupted by cable failures.

This was very time consuming therefore some non-critical aforementioned functions couldn't be implemented due to the lack of time. As I have mentioned it in the description of the implementation section, there is a better API to get movement information faster and accurately, namely the Activity Recognition Sampling API in which we can specify the desired notification interval in concerned with battery life. More frequent notification means worse battery life but more accuracy.

The application in its current form can perform what it's having to do though if the screen isn't lit than it would take 1-2 minutes to the resting device to notify the application. The implementation of the functions mentioned in the further development section of the thesis would mean an application that can change our perception of parking in which it wouldn't require the user's constant active actions to manage the parking.

11 ÁBRAJEGYZÉK

12 IRODALOMJEGYZÉK

[1] (https://www.csmagics.com/single-post/2013/07/24/this-is-the-title-of-your-first-image-post), utoljára megtekintve: 2021. 05. 16.
[2] (https://mobisoftinfotech.com/resources/blog/cross-platform-app-development/), utoljára megtekintve: 2021. 05. 16.
(https://blog.logrocket.com/flutter-vs-react-native-vs-xamarin/), utoljára megtekintve: 2021. 05. 16.
(https://promatics.medium.com/heres-who-would-win-if-xamarin-flutter-and-react-native-fight-out-in-2021-8fa6e22fdfbb), utoljára megtekintve: 2021. 05. 16.
[3] (https://hackernoon.com/xamarin-the-revolution-in-cross-platform-mobile-app-development-123xb3xne), utoljára megtekintve: 2021. 05. 16.
[4] (https://stackoverflow.com/questions/18570878/how-to-know-the-current-os-platform-of-the-executing-code-android-ios), utoljára megtekintve: 2021. 05. 16.
[5] (https://www.nuget.org/packages/UniversalBeaconLibrary/), utoljára megtekintve: 2021. 05. 16.
[6] (https://support.kontakt.io/hc/en-gb/articles/201620741-iBeacon-Parameters-UUID-Major-and-Minor), utoljára megtekintve: 2021. 05. 16.
[7] (https://openuuid.net/signin/), utoljára megtekintve: 2021. 05. 16.
[8] (https://docs.microsoft.com/en-us/xamarin/get-started/quickstarts/database?pivots=windows), utoljára megtekintve: 2021.12.12
[9] (https://developers.google.com/awareness), utoljára megtekintve: 2021. 05. 16.
[10]
(https://developers.google.com/android/reference/com/google/android/gms/awareness/fence/DetectedActivityFence), utoljára megtekintve: 2021. 05. 16.
[11] (https://developer.apple.com/documentation/coremotion#topics), utoljára megtekintve: 2021. 05. 16.
(https://developer.apple.com/documentation/bundleresources/information_property_list/nsmotionusagedescription), utoljára megtekintve: 2021. 05. 16.
[12] (https://forums.xamarin.com/discussion/146848/how-to-check-location-app-permissions-in-xamarin-ios-apps), utoljára megtekintve: 2021. 05. 16.
[13] (https://developer.apple.com/documentation/corelocation), utoljára megtekintve: 2021. 05. 16.
[14] (http://www.davidgyoungtech.com/2019/10/18/permission-denied), utoljára megtekintve: 2021. 05. 16.
[15]
(https://www.csharpcodi.com/vs2/4193/sensus/Sensus.Android/Probes/Movement/AndroidActivityProbe.cs/), utoljára megtekintve: 2022. 01. 19
[16] (https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/), utoljára megtekintve: 2022. 01. 19
[17]
(https://developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognitionClient), utoljára megtekintve: 2022. 05. 15