Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Nagy Kristóf

FELÜGYELT SZÁLLÍTMÁNYOZÁST TÁMOGATÓ IOT RENDSZER

KONZULENSEK

Gulyás György, ügyvezető igazgató (Vitarex Stúdió Kft.)

Jenei Péter (Vitarex Stúdió Kft.)

Dr. Lengyel László, egyetemi docens

BUDAPEST, 2018

Tartalomjegyzék

Összefoglaló **5**

Abstract **6**

1 Bevezetés **7**

2 A szállítmányozás napjainkban **9**

2.1 Jelenleg alkalmazott rendszerek 10

2.2 Az okos szállítmányozási rendszer koncepciója 11

3 Kiberfizikai rendszerek és alkalmazásaik **14**

3.1 Kiberfizikai rendszerek 15

3.2 Internet of Things 15

3.3 Ipar 4.0 az iparosodás új korszaka 17

4 Az okos szállítmányozási architektúra felépítése **18**

4.1 Az eszköz réteg bemutatása 18

4.2 A felhő réteg bemutatása 23

5 A proof of concept rendszer **27**

5.1 A rendszer követelményei 27

5.2 A telematikai eszköz komponensei 29

5.2.1 A felhasznált Raspberry Pi eszköz és tápellátása 30

5.2.2 A felhasznált adatgyűjtő modulok és szenzorok 31

5.2.3 Az eszköz kommunikációs megoldása 33

5.3 A gyakorlati megfontolások a telematikai eszközben 34

5.3.1 Számítási eszköz és tápellátásának megoldása 34

5.3.2 Szenzor modulok megoldása 35

5.3.3 Kommunikációs megoldások 35

5.4 A telematikai eszközön futó kód bemutatása 36

5.4.1 A kód futtató környezetének bemutatása 36

5.4.2 Az adatgyűjtés menete és a gyűjtött adatok értelmezése 37

5.4.3 Az adatok küldése, tárolása és riasztások kezelése 40

5.5 A telematikai eszköz gyakorlati szoftveres megoldásai 41

5.5.1 Cloud to device üzenetek 41

5.6 A felhő réteg szoftveres megoldásai 42

5.6.1 Azure felhőszolgáltatások 43

5.6.2 A felhő rétegen alapuló szolgáltatások 50
5.7 A felhő réteg gyakorlati szoftveres megoldásai 52
5.8 Költség és fogyasztás becslése 54
6 A prototípus rendszer tesztje egy példán keresztül 56
6.1 Eszköz réteg 56
6.2 Felhő réteg 57
6.3 Szolgáltatás réteg 58
6.4 A gyűjtött adatok bemutatása és részletezése 60
7 Konklúzió 64
Irodalomjegyzék 66

HALLGATÓI NYILATKOZAT

Alulírott **Nagy Kristóf**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2018. 05. 17.

...…….
Nagy Kristóf

Összefoglaló

Szakdolgozatomban egy olyan IoT rendszer fejlesztésével foglalkozom, amely képes a közúti szállítmányozás folyamatát átláthatóbbá és nyomon követhetőbbé tenni, ezzel növelve annak hatékonyságát és ellenőrzöttségét.

A szállítmányozás során gyakran merülnek fel olyan problémák, amelyek súlyos kihatással vannak a folyamat hatékonyságára. A késések, elakadások, balesetek például jelentős időbeli veszteséget jelentenek, amely bizonyos szállítmányok vagy a szállítmányt átvevő felek esetében nem elfogadhatóak. Ezen felül a szállítmányt átvevő felek számára szintén kiemelt jelentőséggel bír a megérkezett áru fizikai állapota. Az esetükben például hasznos információval szolgálhat, ha tudják, hogy a szállítmányban nem keletkezett kár a szállítás során.

Az ilyen esetek mind nyomon követhetővé és jelezhetővé válnak a szállítmányozásban résztvevő felek számára egy rendszer segítségével. Ennek a rendszernek működése során adatokat kell gyűjtenie az egyes szállító egységekben elhelyezett adatgyűjtő eszközök segítségével. Ezeket a kigyűjtött adatokat egy felhőszolgáltatás keretein belül fel kell dolgoznia és a feldolgozott adatok alapján különböző szolgáltatásokon keresztül informálnia kell a felhasználókat.

Egy ilyen adatgyűjtő rendszer elkészítésével szeretném a szállítmányozás folyamatát felügyelhetőbbé és ellenőrizhetőbbé tenni.

Abstract

In my thesis I deal with the development of an IoT system, which can make the road transportation process more transparent and traceable, thus increasing its effectiveness and inspectedness.

Problems often arise during the transportation process, which have a serious impact on the efficiency of it. Delays, jams, accidents for example represent significant time losses, which are not acceptable for certain consignments or for the receiving parties of the shipment. In addition, the physical status of the goods arriving are also important to the recipients of the consignments. For example, in their case it is a useful information if they know that the cargo did not suffered any damage during the process of the shipment.

Such cases can be traced and indicated to the shipping parties by using some kind of observer system. This system should work with the data collected from the data collecting devices, which are located in the transportation units, during the process of the transportation. These data should be processed within a cloud service. The system should inform the users based on the processed data via various services.

By creating this data collecting system, I would like to make the process of the road transportation more controllable and more verifiable.

1 Bevezetés

Ahogy körülnézünk a minket körülvevő megállíthatatlanul fejlődő világban, lépten nyomon mindenütt okos eszközökkel és intelligens megoldásokat alkalmazó rendszerekkel találkozunk.

Nincs ez máshogy az iparban sem, ahol az Ipar 4.0 jelentette fejlődési versenyben, a legnagyobbak élen járva az okos gyárak és gyártósorok hálózatának termelésbe állításával, a minél nagyobb hatékonyságra és termelés maximalizálására törekszenek gyártási folyamataik optimalizálásával. Viszont, ahogy az emberi testnek is nélkülözhetetlen a véráram szállította friss oxigén, ezek az okos gyárak és más piaci szereplők is csak félkarú óriások a megfelelő szintű okos nyersanyag és áruszállítás hiányában. Egy ilyen szintű okos szolgáltatás kialakításának több elvárásnak és szempontnak is meg kell felelnie.

Például a szállítmányozás során kitüntetett szerepe van az időnek, mint szállítási tényezőnek. Az áruházak, gyárak és más piaci szereplők számára fontos, hogy pontosan tudják a szállító egység érkezésének idejét, hogy a megfelelő erőforrásaik készen álljanak az áru gördülékeny rakodására, illetve átvételére. Továbbá hasznos információval szolgálhat, ha időben értesülnek az érkező szállító egység elakadásáról, legyen szó közúti balesetről, torlódásról vagy késésről. Ezen felül a jelenlegi szállítmányozás egy további nagy problémája, hogy nincs semmiféle biztos információ arról, hogy a szállítmányt megfelelő módon kezelték-e az út során, abban nem keletkezett-e kár. Ilyen esetekben, ha a szállítmányozás útja során az árut érő fizikai hatásokról mért információval rendelkezünk, bizonyíthatóvá válik a szállítás folyamatának helyessége, illetve az, hogy nem keletkezett kár az áruban.

Ezért szakdolgozatom céljául tűztem ki egy olyan okos szállítmányozási rendszer architektúrájának és ötletének kialakítását, amely alkalmas a felsorolt esetek jelentette szállítmányozási problémák és igények kiszolgálására. Ennek kialakításához a napjainkban is ismert és használatos kiberfizikai és IoT rendszerek felépítését vettem alapul.

Ennek megfelelően a szállítási folyamat kiszámíthatóságának érdekében a fizikai eszközök szintjén valós idejű monitorozást valósítunk meg az egyes szállítójárművek

útjai során. A raktérben, vagy arra alkalmas helyen fix pozícióban elhelyezett eszközhálózat segítségével, a szállítmányozás során keletkező és a szállítmányra jellemző adatokat gyűjtünk szenzorok segítségével. Az alkalmazott szenzorok között vannak általános jellegű információt közlők, mint például a GPS modulok vagy a gyorsulás mérő szenzorok és vannak a szállítmány jellegéből kifolyólag specifikus szenzorok. Például hűtött áru esetén hőmérséklet mérő szenzorok segítségével gyűjthetünk adatokat. Élő állatokat feltételező szállítmány esetén oxigén szint mérésére alkalmas szenzorral végezhetünk adatgyűjtést. Gázolaj és egyéb tartályban szállított áru esetén a tartályban nyomás szint mérésére alkalma szenzort alkalmazhatunk. Ezeket a mért szenzor adatokat a már kiépített mobilhálózaton keresztül valamilyen rádiós technológiát alkalmazva továbbítjuk egy központi felhőszolgáltatásba.

Itt aztán megkezdődik a beérkező adatok feldolgozása. Az adatok különböző adatfolyamokba kerülnek át útválasztók segítségével attól függően, hogy mi fog velük történni. Minden beérkező adat egy feldolgozást követően naplózásra kerül egy adatbázisba. Az adatbázisba került adatok később visszakereshetőek és feldolgozhatóak igény szerint. Egy másik adat útvonal mentén a beérkező adatokon valós idejű feldolgozást végzünk. A feldolgozás eredményéről egy grafikus felületen keresztül informáljuk a felhasználóinkat az aktuálisan mért szenzor értékekről különböző grafikonok és diagrammok segítségével. Szintén a felhasználók informálásának céljából, a beérkező adatokat megvizsgáljuk, hogy tartalmaznak-e olyan értékeket, amelyek a szállítás során fellépő rendellenességre utalnak, például túl magas hőmérsékletre, nyomásra, rázkódásra vagy elakadásra. Ha ilyen esetet detektáltunk akkor a megadott felhasználókat értesítjük a történtekről például egy e-mailes üzenetben. Továbbá az adatokból készíthető elektronikus jegyzőkönyv egy kiszámíthatóbb, megbízhatóbb kapcsolat kiépítését teszi lehetővé a szállítmányozási szolgáltatók és a felhasználóik között.

Ennek tekintetében az általam elképzelt rendszer IoT megoldásokat felhasználva, alkalmas a szállítmányozás folyamatának felügyeletére és támogatására. Ezen felül növeli az szállítmányozási folyamat kiszámíthatóságát, valamint elősegíti az áruszállításban érdekelt felek közötti kölcsönös bizalom kiépítését, hiszen a szállítás folyamatának minden mozzanatáról bizonyítható információval rendelkeznek.

2 A szállítmányozás napjainkban

Mai modern világunk egyik legérzékenyebb pontja, valamint mozgató rugója a szállítmányozás, még ha ezt nem is mindig vesszük észre. Az áruk, termékek, félkész termékek és nyersanyagok egyik pontból a másikba időben történő eljuttatása, kiemelt jelentőséggel bír minden termelési láncban.

A szállítmányozás a mai értelemben véve az árutovábbítás folyamatának megszervezése, amelynek célja a küldemény lehető legkisebb költséggel, legbiztonságosabban és a megfelelő időben történő optimális célba juttatása a rendeltetési helyre. Azt a természetes vagy jogi személyt, aki az áru továbbításával, azaz elfuvarozásával egy másik vállalkozót bíz meg díjazás ellenében, fuvaroztatónak, a megbízás megtörténtét és a fuvarozási feladat elvállalását tanúsító okmányt pedig fuvarozási szerződésnek nevezzük. A szállításon ennek a fuvarozási ügyletnek a teljesítését értjük. Itthon Magyarországon, 1989-ben öt társaság (MASPED, Raabersped, Voláncamion, Hungarocargo, Technosped) létrehozta a Magyar Szállítmányozói és Logisztikai Szolgáltatók Szövetségét, amely ma közel száz Magyarországon bejegyzett társaság szakmai érdekképviselete [1].

A szállítmányozás a következő részterületekre osztható hazánkban:

- Vasúti szállítmányozás
- Közúti szállítmányozás
- Folyami szállítmányozás
- Légi szállítmányozás
- Tengeri szállítmányozás

Ezek közül a szakdolgozatom témájának középpontjában a közúti szállítmányozás áll, amely a szállítmányozás szektorain belül azért kiemelkedő mert a belföldi viszonylatban a 2017-es évre vonatkozóan elérte az árutömeg szállítás terén a 81% -os részesedést [2]. Ebből is látszik, hogy a gyárak, még inkább az okos gyárak nagymértékben ki vannak szolgáltatva a szállítmányozás folyamatának, minőségének. Ezért elengedhetetlennek tartom a szállítmányozás megfelelő mértékben történő korszerűsítését, felzárkóztatását az okos gyárak technológiai, szervezési szintjéhez.

2.1 Jelenleg alkalmazott rendszerek

Mielőtt nekivágtam volna a saját rendszerem megalkotásának, fontosnak tartottam, hogy tájékozódjak a már létező rokon felhasználású szolgáltatásokról, azok hiányosságairól és erősségeiről.

A hazai és világpiaci viszonylatban is több olyan innovációs és okos informatikai megoldás található, amelyek a közúti fuvarozást hivatottak valamilyen szempontból kiszámíthatóbbá, monitorozhatóbbá tenni. Ezen szolgáltatások többsége egyéni megoldás, nem egy egységes szabványosított koncepció része. Minden nagyobb flottát kezelő fuvarozó cég használ valamilyen mértékben pozíció, illetve helymeghatározáson alapuló technológiákat, amelyekkel saját működésüket kívánják hatékonyabbá tenni. A GPS nyomkövetésen túlmenő innovációs törekvés három irányzatát véltem felfedezni.

Az első az úgynevezett „platooning”[3], amely összekapcsolt tehergépjárművek hálózatát és hálózatban történő önvezető mozgását teszi lehetővé osztagok formájában. Ennek az alapja a következő technológia, amely a „machine to machine” kommunikáció megvalósítására keres a szállítmányozáson belül megoldást. Hasonlóan az önvezető autók kooperációs közlekedési mintájához, itt is a szomszédos tehergépjárművek együttműködésére keresnek megoldást. Az általam elképzelt szolgáltatással legtöbb közös vonást hordozó jegyeket az úgynevezett szállítmányozási telematikai rendszereknél véltem felfedezni. A név a működésből eredeztethető, mivel telekommunikációs és informatikai megoldásokat tartalmaz a rendszer. Ezek a koncepciók alkalmasak az árukat és az azokat mozgató eszközök nyomon követésére, beleértve a navigációs és a szállítmány jellegére vonatkozó adatokat is. Emellett folyamatos adat és kommunikációs kapcsolatot biztosítanak a jármű vezetőjével. A hazai piacon a Waberer’s – Szemerey Logisztikai Kft. alkalmazott először ilyen jellegű rendszert [4]. Ezek bevezetése hosszadalmas és sok esetben képzettebb, tapasztaltabb munkaerő alkalmazását követelik meg, amely a fizetendő bérek emelkedésével jár.

Egyes elképzelések szerint a szállítójárműveket 2022-re szabványosított telematikai rendszerekkel szerelik fel, amely nagy lehetőség biztosít a hasonló koncepciókat alkalmazó szolgáltatások számára. Ezért indultam el én is egy saját rendszer ötletének megvalósításával, amelynek központjában a szolgáltatók és felhasználóik közötti nyílt kapcsolat megteremtése áll.

2.2 Az okos szállítmányozási rendszer koncepciója

A szállítmányozás okossá tételéhez egy olyan rendszert kell létrehoznunk, amely által nyújtott szolgáltatás eleget tesz az általunk támasztott és elvárt követelményeknek. Ehhez alapvetően a jelenlegi rendszerek problémáira, nehézségeire és megoldatlan kérdéseire szükséges fókuszálni, ezekre kell megoldást találni.

Tehát ahhoz, hogy megfelelő képet kell alkossunk az elvárásainkról, ismernünk kell a rendszerbe felhasználóként becsatlakozók igényeit, illetve tisztában kell lennünk a jelenlegi rendszerek hiányosságaival is. Például a szállítmányozást kezdeményező fél számára fontos ismerni a szállítás során, annak természetéből adódóan az árut érő fizikai hatásokat, hogy azt a szállításnak megfelelően csomagolja, készítse elő. Továbbá fontos lehet neki a szállító jármű holléte, hogy a szállításra váró áru időben útnak indulhasson, illetve megérkezhessen.

A szállítmányozó cég számára célszerű ismernie a szállíttatók igényeit, hogy a flotta szervezés során optimális legyen a járműállomány kihasználtsága, és minden áruhoz az annak legmegfelelőbb típusú szállító egység kerüljön. Többek között fontos, hogy ellenőrizhető legyen a szállítás során az áruk fizikai állapota, a jármű pozíciója, a szállítmányhoz való hozzáférések körülménye, az esetlegesen fellépő nem várt jelenségek, mint balesetek, közlekedési torlódások, vagy a szállítmány természetéből adódó egyéb kulcsfontosságú esetek, amikor egy riasztásnak vagy időben jelzett eseménynek köszönhetően beavatkozhatunk, ezzel is növelve a szállítás minőségét.

A szállítmányt átvevő fél számára kardinális kérdés lehet a szállító egység pozíciója és a szállítmány várható érkezése, ha például ezen múlik egy gyártó sor megakadása, amely komoly termelés- és bevételkiesést jelenthet. Ezen felül szintén érdekelt a szállított áru épségben történő átvételében és az árut a szállítás közben ért esetleges fizikai hatások ismeretében. Ezen információk tudatban például egy okos gyár gyártósorainak ütemezése könnyen igazítható a feldolgozásra érkező alapanyagok érkezési sorrendjéhez, ritmusához és így akár leállás mentesen, de mindenképpen optimalizálhatóan szervezhető a munkafolyamat.

A szállítmányozásban alapvetően a fent említett három érintett fél elvárásait tekinthetjük meghatározónak, de elképzelhetők még egyéb közvetetten felmerülő igények is, mint a szállítás útvonalának karbantartójának igényei, akinek fontos lehet tudnia, ha például veszélyes anyag került az útra a szállítmányból egy baleset során.

Az elvárásaink ismeretében alkothatunk egy olyan egységes rendszert, amely segítségével a mainál okosabb, magasabb minőségi szintű szállítmányozást valósíthatunk meg. Ehhez adatokat kell gyűjtenünk a szállítmányozás folyamatáról, amelyekből információt előállítva eleget tehetünk a felmerülő elvárásoknak és kiszolgálhatjuk a jelentkező igényeket. Az általam elképzelt rendszer egy olyan szállítmányozást támogató IoT megoldás, amely segítségével a szállításban érdekelt felek, a számukra lényeges információk tudatában körültekintőbben hozhatnak fontos döntéseket. A szolgáltatás alapvetően magát a szállítmányozó cégeket célozza meg, ugyanis az általuk kezelt flotta egyes eszközeiről szeretnénk adatot gyűjteni a szállítmányozás során, és felhasználni az ezekből az adatokból előállított információt. Ehhez a raktérben egy adat gyűjtő egységet, eszközt helyezünk el, amely a szállított áru profiljából következően, a megfelelő szenzorok segítségével monitorozza az eseményeket. Ilyen mérés lehet például élőállat esetén az oxigén szint, tartályok esetén a nyomás vagy hűtött áru esetén a hőmérséklet, ezekre az esetekre később bővebben is kitérek. Az egyes járművekben elhelyezett adatgyűjtő egységekből az adatok, internet hozzáférést feltételező kapcsolat mellett egy központi felhő szolgáltatásba kerülnek továbbításra.

A felhő által megvalósított szolgáltatás keretein belül a beérkezett adatok feldolgozásra kerülnek. Ezen szolgáltatás célja többek között:

- az adatok szűrése és kategorizálása
- az adatok tárolása és naplózása
- az adatok big data elemzése
- az egyes szállítmányok és jellemzőik folyamatos nyomon követése és megfigyelése
- a rendkívüli esetek jelzése és előrejelzése
- a szállításban érdekelt felek informálása
- valós idejű állapot közvetítése
- a szerződő felek közötti bizalom és hitelesség erősítése

Természetesen ezen felül az adatok felhasználásának számtalan módja elképzelhető.

Egy ilyen szolgáltatás segítségével a szállításban érdekelt felek folyamatosan nyomon követhetik az áru pontos fizikai helyzetét, állapotát és az árut ért minden lényeges fizikai hatást. Ennek tükrében biztosak lehetnek abban, hogy az árut a megfelelő módon kezelték a szállítás során, vagy ha éppen valami rendellenesség lépett fel annak idejét és körülményeit is pontosan tudni fogják.

A szolgáltatás keretein belül keletkező adathalmaz átadása (nevezzük ezt elektronikus naplónak) növeli a szerződött felek közötti bizalmat és nyomon követhetőséget. Támogatja továbbá a rendkívüli vagy hirtelen fellépő esetekben a hatékony és gyors döntéshozatalt. Például egy kulcsfontosságú szállítmány elakadása esetén ezt jelezve, a gyár képes lassítani a munkafolyamatot, hogy ne kelljen leállítani a termelést az elakadt szállítmány megérkezéséig.

A szállítmányozó cég a gyűjtött adatokból további kutatásokat végezhet, például az egyes járművek rázkódás idősorából hosszú távon előre jelezhető az alkatrészek kopása vagy a sofőrök vezetési stílusa, modora.

A teljes működő rendszer elkészítése jelentős erőforrásokat és időt venne igénybe, ezért a munkám során ennek csak egy kísérleti jellegű projektjét készítettem el. Ennek a projektnek a célja alátámasztani, hogy az elképzelt rendszer nyújtotta szolgáltatás működőképes és megvalósítható az elképzeléseim alapján. Tehát képesek vagyunk a mozgó járművekből adatot gyűjteni, az adatot a felhőszolgáltatásba továbbítani, az adatokat itt feldolgozni és a feldolgozott adatok alapján különböző információkkal ellátni a felhasználókat.

A projekt elkészítésének kezdetétől fogva ügyeltem arra, hogy a megoldásokat, amelyeket felhasználtam, miként lenne célszerű átültetni, megvalósítani egy vállalati környezetben működő rendszerbe. Ennek azért szenteltem különös figyelmet, mivel a projekt önmagában csak akkor hasznos, ha útmutatást ad egy elkészítendő éles rendszer megoldásaihoz.

Így az architektúra megtervezése során, a fizikai eszközöktől kezdve, az igénybe vehető felhőszolgáltatásokon keresztül, egészen az eszközök tápellátásáig szem előtt tartottam a dinamikusan skálázható megoldások használatát, illetve megemlítését. Mielőtt még ezekre részletesebben is kitérnék szeretném bemutatni az általam alkalmazott architektúra alapjait és hogy ennek létrejöttét, milyen egyéb tényezők befolyásolták.

3 Kiberfizikai rendszerek és alkalmazásaik

Ha szeretnénk pontosabban is megérteni a napjainkban éppen zajló ipari átalakulás folyamatában alkalmazott rendszerek működését, érdemes egy kicsit az elmúlt évek, évtizedek technológiai fejlődésének néhány meghatározó pontjára rávilágítani. Ezek a technológiai megoldások mind alapját képezik annak az architektúrális megoldásnak, amelyet az általam megemlített rendszerek és én is felhasználok a saját munkám során.

Fontossági sorrendben haladva elsőként, nélkülözhetetlennek tartom a mikroelektronika közel exponenciális évenkénti fejlődését, amelynek köszönhetően a 2017-es évben már mm^2 -enként 25 millió tranzisztort is képesek vagyunk integrált áramkörökben elhelyezni. Ennek köszönhetően az eszközök fizikai mérete is elért egy olyan szintet, amely lehetővé teszi az emberi mindennapokban történő, szinte észrevétlen alkalmazását. A méret csökkenésével párhuzamosan, a gyártás fejlődésének eredményeként a piaci árak elérték azt a szintet, ahol az eszközök nagy számban történő felhasználása már nem ütközik anyagi keretekbe [5].

Szintén meghatározó szerepet tulajdonítok a rádiós és távközlési diszciplínák fejlődésének, amelyek eredményeként eszközeink, technológiák és protokollok széles skáláját alkalmazva, képesek a kommunikáció legkülönfélébb és az adott kommunikációs probléma helyzetéből kifolyólag a legkézenfekvőbb módon kapcsolatot létesíteni. Hálózataink és a hálózati megoldások fejlettsége elért egy olyan szintet, amely már képes lépést tartani a korábban említett nagyszámú eszközök közti információ cserével. Továbbá a napjainkban kialakuló félben lévő 5G szolgáltatás várhatóan még inkább alkalmas lesz az eszközeink jelentette kommunikációs elvárásaink teljesítésére. Ezen megoldások és protokollok az elmúlt évtizedek mérnökeinek kitartó munkáját és kreativitását dicsérik.

Természetesen a fent említett eredményeken túl számtalan egyéb megoldás és mérnöki megfontolás játszott szerepet a mai rendszereink kialakításában, amelyeket most nem emeltem ki, de szerves részét képezik a most következő architektúráknak.

3.1 Kiberfizikai rendszerek

A kiberfizikai rendszerek (angolul „cyber-physical system“, CPS) alatt olyan informatikai, szoftvertechnológiai, valamint mechanikai- és elektronikai elemek egységbe kapcsolását értjük, ahol az elemek egy „adat-infrastruktúrán” keresztül kommunikálnak egymással.

Ezen beágyazott érzékelőkkel és beavatkozókkal rendelkező rendszerek működésük során monitorozzák a világot és intelligens, megbízható módon beavatkoznak a környezetükbe, a céljuk elérésének érdekében. Egy kiberfizikai rendszer gyakran egy szabályozási kört valósít meg, ezért magas szintű együttműködést követelünk meg a rendszer fizikai és számítási elemei között. Továbbá, ezen rendszerek többsége internethez integrált vagy valamilyen rádiós technológiát alkalmaz a kommunikáció során, így képesnek kell lenniük az interneten keresztül elérhető tudás és szolgáltatások, felhasználói igények által vezérelt befogadására, beleértve a felhők által biztosított számítási lehetőségeket is [6]. Ezek a rendszerek egyre nagyobb mértékben válnak mindennapi életünk részévé és minden területen megtalálhatóak lesznek, ahol beépített intelligenciára, távoli beavatkozásra vagy eszközök magas szintű együttműködésének vezérlésére lehet szükség.

A rendszer felépítését tekintve legalacsonyabb szinten a fizikai környezetbe ágyazott mikrokontrollerjeinken, illetve a hozzájuk csatolt szenzorokon és beavatkozókon keresztül monitorozhatjuk a világunk rendszerezni kívánt részét. Ezek a beágyazott eszközök valamilyen vezetékes, vagy vezeték nélküli kommunikációs hálózaton keresztül összeköttetésben állnak egymással, vagy egy nagyobb számítási kapacitással rendelkező eszközzel, amelyekkel egy meghatározott protokollt követve történik az információ csere. Az információ közlésen és a felügyeleten túl a hálózaton történő kommunikációra a fizikai környezetből adódó méretbeli alkalmatlanság, távolság vagy dinamikus helyzet változtatás miatt van szükség. Sok esetben a rendszer részét képező nagy számítási kapacitással rendelkező eszközök méretüknél fogva nem alkalmasak a közvetlen helyszínen történő alkalmazásra, ezért vezetékes vagy vezeték nélküli, rádiós technológiákat alkalmazva érjük el őket.

3.2 Internet of Things

A dolgok internete (angolul „Internet of Things“, IoT) az okos eszközök (beágyazott érzékelőik és beavatkozóik), járművek, épületek interneten keresztül

egymással történő hálózatba kötése, a kinyerhető információk cseréjének és gyűjtésének érdekében [7].

Ezen technológia a tágabb értelemben vett kiberfizikai rendszereknek egy részét képezi. Az IoT azonban néhány dolgot illetően erősen eltér a korábban említett kiberfizikai rendszerektől, amely tisztázásra szorul: A kiberfizikai rendszerekkel ellentétben a feladatok jelentős része az adatok kinyerésére, gyűjtésére, monitorozására, elemzésére és tárolására korlátozódik. Tehát a rendszereinktől nem feltétlenül várunk el, de nem is zárjuk ki az intelligensnek vélt beavatkozást. Általában inkább az emberi közbelépés, beavatkozás jellemző, ha arra éppen a kinyert adatok alapján szükség lehet.

Egy másik fontos jellemzője az IoT-nak, hogy kifejezetten az IP (Internet Protocol) képes eszközök összeköttetésén alapul. Ennek köszönhetően például az IP képes eszközök száma 2016-tól 2017-ig 31%-kal növekedett, így ma körülbelül 8.4 milliárd eszközünk csatlakozik az internetre. Némi aggodalomra adhat okot az a becslés, miszerint 2020-ra ez a szám elérheti a 30 milliárdot, amely nem csak nagymértékű infrastrukturális terhelést jelentene, de az adatbiztonsági és adatvédelmi szempontból amúgy is sebezhető rendszert még kiszolgáltatottabbá tenné. Mindezek ellenére az élet szinte minden területén jelennek meg különböző megoldások, amelyek alapvető célja az eddig ember által végzett tevékenységek optimalizálása, a kényelem maximalizálása, illetve a korábban erőforrás hiányában elvégezhetetlen feladatok végrehajtása.

Alapvetően három, alkalmazásában elkülönülő területet szoktunk megkülönböztetni az IoT-ban.

Fogyasztói IoT (Consumer IoT)

A fogyasztói (Consumer IoT) alkalmazások alapvetően magánszemélyek igényeit szolgálják ki. Ilyenek például az okos otthon (Smart Home) és egyéb otthon automatizáló rendszerek, a viselhető vagy egészségügyi eszközök, de akár a robot porszívók, hűtők és egyéb okosnak nevezett háztartási eszközök is.

Kereskedelmi IoT (Commercial IoT)

A kereskedelmi szinten az egyének helyett inkább csoportosulások, szervezetek követelményeit szolgálják ki ezek a rendszerek. Ide tartoznak például a különböző energia menedzsmenttel kapcsolatos rendszerek, mint az okos világítás vagy

áramellátási rendszerek, víz és gázvezeték rendszerek, az okos parkoló rendszerek, az intelligens forgalom irányítás, közlekedés és sorolhatnánk még a közösségi felhasználási lehetőségeket.

Ipari IoT (Industrial IoT**)**

Az ipari IoT rendszerek alapjában véve cégek, vállalatok szükségleteit hivatottak kielégíteni. A gyártósorok és működésük automatizálása, optimalizálása további termelékenység, precizitás és hatékonyság növekedéssel kecsegtet, amelyek haszna a nyereségben is megmutatkozhat. Ezért egyre több helyen jönnek létre okos, önműködő gyártó sorok, raktár rendszerek, mezőgazdasági öntöző és állattartó rendszerek, amelyek minimális emberi beavatkozással képesek működni. Ezen rendszerek egy új ipari forradalom előhírnökei.

3.3 Ipar 4.0 az iparosodás új korszaka

Az emberiség eddig három ipari forradalmat élt meg. A gőzgépeket, futószalagokat és az automatizációt követően, most egy teljesen új, negyedik ipari forradalom zajlik. A forradalom keretein belül a fizikai gépek és tárgyak egy okos hálózatba kapcsolódnak, ezáltal a gazdaság egyetlen hatalmas, intelligens információs rendszerbe integrálódik [8]. Ezekben az okos gyárakban a moduláris szerelősorokon futó folyamatokat lebonyolító kiberfizikai rendszerek összeköttetésben állnak egymással a nagyobb hatékonyságú közös munka érdekében. Monitorozzák a munkát, döntéseket hoznak, jeleznek egymásnak, jelezni tudják a hibákat és azt is előre tudják jelezni, ha például egy alkatrész elhasználódik, így időben megoldható annak cseréje a gyártósorok leállítása nélkül [9].

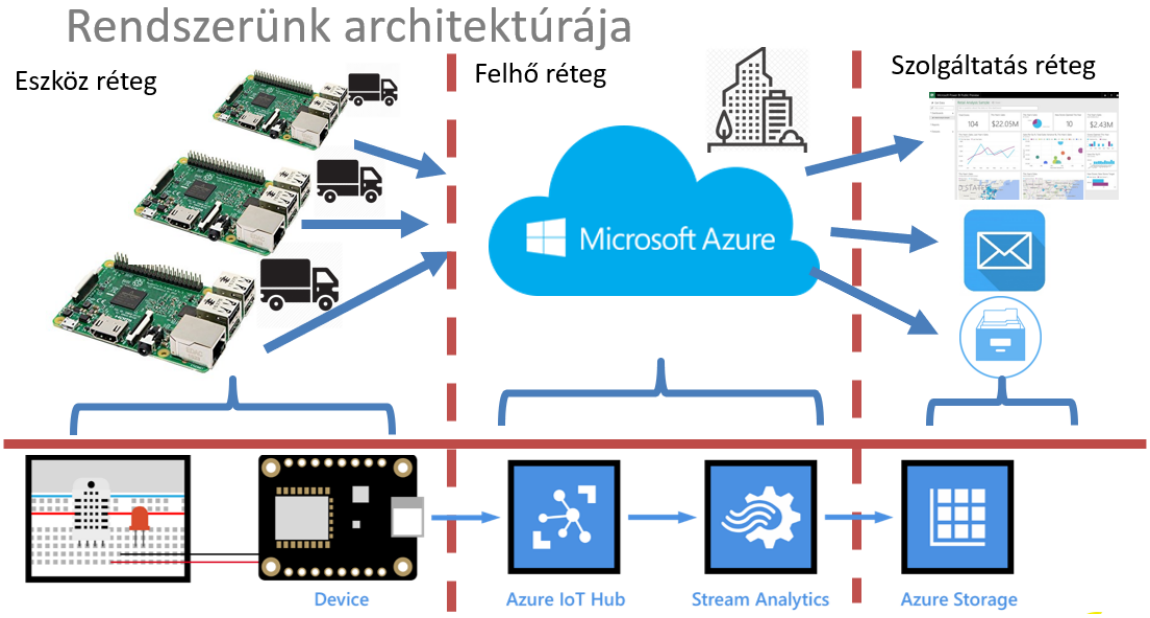
Az így működő okos gyárak az iparosodás egy új korszakát hozzák el, ahol már nem emberek, hanem gépek irányítják az automatikus gyártást és a működtetéshez emberi beavatkozás csupán a legmagasabb szinten szükséges. Mindazonáltal ezeknek az okos gyáraknak is szükségük van alapanyagokra vagy félkész termékekre, amelyeket feldolgozhatnak. Így válik a szállítmányozás az Ipar 4.0 ütőerévé.

Ennek vonzatában az általam elképzelt okos szállítmányozási rendszer Ipar 4.0 megoldásnak tekinthető, mivel annak folyamataival szoros kapcsolatban áll és az alkalmazott architektúrális elemekben is hasonló megoldásokat használ fel.

4 Az okos szállítmányozási architektúra felépítése

Ebben a fejezetben kitérek az általam elképzelt és megvalósított rendszer elkészítése során felhasznált olyan architektúrális elemek ismertetésére, amelyek szorosan kapcsolódnak a problémakörhöz illetve részét képezik a megoldásomnak.

Szisztematikusan először a rendszer mozgó, szállító járműre szerelt komponenseihez kapcsolódó ismerettel kezdem, majd beszélek a háttérben a felhő oldalon történő adatfeldolgozást bemutató megoldásokról és azok szolgáltatásairól. A rendszerem architektúrális felépítését az általam elkészített 4.1-es ábra írja le. Ennek mentén ismertetem az általam felhasznált megoldásokat.



4.1. ábra: A rendszerem architektúrális felépítése

4.1 Az eszköz réteg bemutatása

Az eszköz réteg alatt azt a telematikai egységet értem, amely a szállító jármű rakterében kerül elhelyezésre. Ez az eszköz megfelelő fizikai védőburkolással ellátva és biztonságos, fix helyre beszerelve utazik a rakománnyal. Az eszköz elsődleges feladata a hozzá csatolt szenzorokon keresztül az adatok gyűjtése és továbbítása feldolgozásra. Ehhez viszont érdemes alaposan, körültekintően megválasztanunk ezt az eszközt.

Ha az alapvető számítási és adatműveleti feladatokat szeretnénk egy korlátozott méretű eszközzel elvégezni, ma már egykártyás számítógépek (Single Board Computer)

nagy választékából szemezgethetünk, annak érdekében, hogy a nekünk legmegfelelőbbet alkalmazhassuk. Ezek az eszközök szinte teljes értékű számítógépként funkcionálnak, viszont némi teljesítménybéli visszaesésért cserébe elférnek egy közel bankkártya méretű nyomtatott áramköri lapon. Ezek a számítógépek ugyanúgy rendelkeznek mikroprocesszorral, memóriával, ki és bemeneti csatlakozókkal, mint hagyományos asztali társaik. További előnyük a méretük mellett, a relatív olcsónak mondható piaci áruk, így nagy számban alkalmazhatóak ezek az eszközök különféle IoT rendszerek csomópontjaiként.

Ahogy említettem, a kereskedésekben rengeteg ilyen - eltérő funkcionalitású és teljesítményű - eszköz kapható más és más feladatokra, különböző ár-érték arányban. Az általam választott eszköz és gyártója, ár-érték arányban közel a legjobb kínálatot nyújtja, mindamellett, hogy a piacon ők rendelkeznek a legtöbb tapasztalattal és élenjárók a technológiában is. Tehát a választásom az általam használt egykártyás számítógépre, a Raspberry Pi 3 Model B-re esett [10]. Mindenesetre szerepeljen még itt néhány említésre méltó konstrukció, amelyeket szintén használhattam volna munkám során: Banana Pi M2, Beagle Bone Green, Orange Pi PC, Lattepanda, Raspberry Pi Zero W.

A Raspberrryt alapvetően oktatási célokra fejlesztették ki az Egyesült Királyságban, de ma már számos alkalmazása létezik: különböző szervereken túl, használják még média szolgáltatóként és egyéb automatizáló megoldásokban. A specifikációt figyelembe véve a legfontosabbakat emelném csak ki, mint a 4 magos 1.2 GHz órajelű processzort, 1 GB SDRAM, Wifi, Bluetooth Low Energy, 40db GPIO pint. Az eszköz tápellátása kérdéses volt mivel mozgathatónak kellett lennie nem csatlakozhatott egyszerűen az elektromos hálózatra. A szolgáltatás működése közben a mozgó tápellátást egy hordozható külső akkumulátorral oldottam meg, az otthon történő fejlesztés ideje alatt pedig normál hálózati tápellátást használtam.

A Raspberry eszköz különböző Linux-disztribúciókkal működtethető, illetve a Microsoft is fejlesztett egy Windows 10 IoT Core nevű operációs rendszert, amelyet kipróbáltam jómagam is, de nem találtam túlságosan célravezetőnek. Ezért a munkám során az eszközön a Raspbian operációs rendszert használtam, amely az ismert Debian Linux-disztribúció egy Raspberry Pi-re optimalizált változata. Az operációs rendszer telepítését követően az elengedhetetlen hálózati védelmi lépéseknek jártam utána és arról gondoskodtam, hogy az eszköz alapvetően biztonságban legyen. Később a

fejlesztést VNC távoli hozzáférést biztosító szolgáltatás keretein belül oldottam meg. A Linux-alapú operációs rendszer használatában nagy segítségre voltak a korábbi tapasztalataim és tanulmányaim.

A kívánt adatok gyűjtése érdekében a Raspberry Pi-t külső szenzorokkal kellett összekötnöm. A szenzorok eszközhöz való illesztéséhez a GPIO pin-eket használtam, ezeken keresztül olvastam az aktuálisan mérhető értékeket. Az egyes GPIO pin-ek külön funkcionalitással rendelkeznek. Vannak 3,3V illetve 5,0V feszültségű tápellátást szolgáltató pinek, illetve földeléshez használatosak. Elkülönített pinek tartoznak a I2C, UART illetve soros módon történő adat olvasáshoz és kommunikációhoz. A továbbiakban a felhasznált, illetve a felhasználható szenzorokat szeretném részletesebben ismertetni.

Az adatok gyűjtéséhez rengeteg különféle szenzort alkalmazhatunk. Ezek egy része általánosan minden típusú szállítmány esetén alkalmazható, de vannak olyan érzékelők, amelyeket csak speciális szállítmányok esetén érdemes alkalmazni, hogy néhányat említsek a különböző típusú szállítmányok közül: mint hűtőfuvarozás, élőállat szállítás, veszélyes anyagok szállítása, kőolaj szállítás, pénz szállítás. Így hát magától értetődő, hogy különféle szenzor válogatásra van szükségünk mindegyik előbb felsorolt szállítmány eset kezelésére.

Az általam felhasznált és kipróbált szenzorok többsége egy általános, a szállítmány típusától független célú alkalmazást tesz lehetővé, de röviden megemlítek olyan szenzorokat is, amelyek használatát különböző speciális esetekben tartom célravezetőnek.

GPS modul

A szállítmány pozícióját tekintve elengedhetetlen a globális helymeghatározó rendszer használata. Ezen adatokból képesek vagyunk megállapítani az eszköz földrajzi helyzetét. Az általam kipróbált és felhasznált GY-NEO6MV2 GPS modul, egy repülésvezérlőhöz alkalmazott önálló GPS jelvevő egység, amely külön kerámia borításos antennával rendelkezik az erősebb jel érzékelés céljából. Az eszköz működéséhez 3V - 5V közötti feszültség szükséges, és az alapértelmezett jelolvasáshoz 9600-as baudrate-et használ. A Raspberry UART kommunikációhoz rendelt GPIO lábain keresztül olvastam adatokat a szenzorról.

A különböző GPS protokollokon keresztül közölt változatos adatokból kedvünkre választhatunk. Talán a legfontosabbakat említve: a szélességi és hosszúsági körön mért koordináta, a tengerszint feletti magasság, sebesség, idő és egyéb, a mérések pontosságát jelző adatok, amelyekre később még pontosan kitérek. Ezt a modult kipróbáltam és az általam elkészített rendszerben is alkalmazom.

GY-87 modul

Ez egy 10 szabadság fokú, 3 az 1-ben szenzor modul, amely tartalmaz egy MPU6050- es szenzort, egy BMP180-as szenzort és egy HMC5883L-es szenzort. A modul I2C kommunikációt használ így a Raspberrynek az erre a célra kinevezett GPIO lábain keresztül kommunikálok a szenzor modullal.

Az MPU6050-es szenzor egy 3 tengelyes (X, Y, Z) gyorsulásmérő és giroszkóp. A felvett tápfeszültsége 2,4V-3,45V közé esik, energia igénye 100 µA, a válaszideje 6 ms. A giroszkóp mérési tartománya 250-től 2000 fok/másodpercig terjed mind három tengelyen és az irány érzékenysége 1 fok. A gyorsulásmérő 2-től 16 G -ig mér, szintén mindhárom tengely mentén. Ezen felül rendelkezik egy beépített hőmérséklet szenzorral, amely az adatok kompenzációjához szükséges.

A BMP180-as szenzor egy nyomásmérő egység, amely által felvett tápfeszültség 1,8V- 3,6V közé esik, áram felvétele működés közben 32µA - 650µA között van. A mért nyomástartomány 320 – 1100 hPa közé esik, amely a tengerszinthez képesti -500 métertől a 9000 méteres magasságig terjed, a mérés érzékenysége pedig 0,6hPa, amely nagyjából 0,5 méteres magasság különbségnek felel meg. A fokozott zajvédelem érdekében fémtokkal van ellátva. A nyomásértékek mérése mellett képes még hőmérséklet adatokat is mérni -40-től +80°-ig terjedő sávban, 1 fokos mérési pontossággal. A szenzor válaszideje 5ms.

A HMC5883L-es szenzor egy digitális iránytű, amely lényegében egy magnetométer. A tápfeszültsége 1,8V- 3,6V közé esik és az áram felvétele működés közben 100µA. A mérési tartománya mindhárom tengelyen 1,3 és 8 Ga közé esik az érzékenysége pedig 5 mGa. Ezt a 3 szenzorból álló modult kipróbáltam és az általam elkészített rendszerben is alkalmazom.

DHT22 szenzor

Ez a szenzor egy pára- és hőmérsékletszenzor, amely ideális ökoház, humán-környezet, melegház, speciális környezet monitorozására. A kommunikációhoz elég egy

tetszőleges GPIO lábról történő digitális adat olvasás, mivel önálló kommunikációs felülete van. A működési feszültségtartománya 3,3V-5V közé esik és az áram felvétele mérés közben 1,5mA. A mérési hőmérséklet-tartománya -40 és +80° közé esik, a mérési pontossága 0.1C°. A páratartalom mérési tartománya 0 - 100%-os relatív páratartalmú levegő mérésére alkalmas, mérési pontossága 0,1%. A szenzor válaszideje 1 másodperc. Ezt a szenzort kipróbáltam és az általam elkészített rendszerben is alkalmazom.

További kipróbált, de nem alkalmazott szenzorok és modulok

Ezeket a szenzorokat első sorban azért nem alkalmaztam, mert az általam elkészített prototípus tesztelése során nem képezték a funkcionális követelmények részét és csak bizonyos szimulált esetekben lett volna jelentőségük. Továbbá a Raspberry modul GPIO pin-jei is véges darabszámú szenzort képesek kiszolgálni egyszerre.

MQ-2 éghető gázok érzékelésére alkalmas szenzor. Ez a szenzor elsősorban robbanékony gázelegyek mérésére készült, ezen gázokra került érzékenyítésre. A specifikus gázok: metán, bután, LPG (autogáz), füst. Az eszköz működési hőmérséklete -20 és +50 fok között hitelesített. Az MQ – X szenzor család különféle gázokhoz ismert változatait is gyártják, amelyek mérése a szállítmányozás során fontos információt biztosíthat. Például élőállat szállítása során fontos az oxigén szint ellenőrzése, gázpalackok szállítása esetén detektálható a szivárgás.

Lángérzékelő, amely tulajdonképpen egy infravörös hullámhosszú érzékelő, szűrőkkel és jelformálással ellátott változata. Ennek segítségével a lángot a 760nm és 1100nm közötti spektrumban érzékeli és jelez, ha lángot észlel. Szintén fontos lehet a szállítás során az esetleges tűzesetek időben történő előrejelzésére.

Fotorezisztor, másnéven fotoellenállás a fényerőt, megvilágítást egyszerű ellenállás-változássá képezi le. A megvilágításmérő széles megvilágítás erősség-tartományban üzemképes, jellemzően a látható fény tartományában. Fényérzékeny anyagok szállítása esetén célszerű lehet a használata, illetve a raktér nyitása során fellépő fényerősség változással detektálhatók az ajtónyitások is.

PIR-szenzor, mozgásérzékelő, amely maximum 5 m távolságról képes érzékelni a változást. Ez teszi alkalmassá otthoni területvédelmi rendszer kiépítésére, de a szállítmányhoz való hozzáférések figyelésére is alkalmas lehet.

Esőszenzor, amely tipikusan nem eső mérésére, de rácseppenő folyadék észlelésére alkalmas lehet. A szenzor működése során a felületére eső nedvesség következtében mérhető feszültség változást tudunk detektálni.

Kamera modul, a Raspberryhez külön megvásárolható modul. Egy mozgás detektálás követően a kamera modult aktiválva felvételt készíthetünk az érzékelés pillanatában, amely így alkalmas lehet a szállítmányhoz történő hozzáférések dokumentálására.

Kapcsolók, segítségével ellenőrizhetővé tehetjük bizonyos feltételek fennállását, mint például a raktér zárásának vagy nyitásának tényét.

Most csak néhány kiemelt fontosságú szenzort soroltam fel, amelyeket rövid kutatás után találtam. Ezen kívül számos szenzor lehet alkalmas és segíthet speciális szállítmányok esetében adatok gyűjtésére.

4.2 A felhő réteg bemutatása

A felhő réteg alatt arra a felhőszolgáltatásra gondolok, amelynek segítségével a beérkező adatokat feldolgozzuk, tároljuk és a felhasználók számára érdemben is felhasználható információt állítunk elő.

A szolgáltatást megvalósító keretrendszer kiválasztása során több lehetőség is felmerült. Ezek közül a legkézenfekvőbbet és az alap szolgáltatásokat tekintve az egyik legmegbízhatóbbat választottam. Emellett konzulensem ajánlása és korábbi tapasztalataim is abban erősítettek meg, hogy a Microsoft kínálta felhőszolgáltatást, az Azure-t válasszam. Az Azure szolgáltatások kínálta lehetőségek egy éles rendszerben is jól skálázódnak, ahol nem csak egy, de akár milliós nagyságrendű végpontot kell kezelni. A szolgáltatásaik felhasználás alapú díjazást követnek, tehát csak a felhasznált számítási erőforrásokért kell fizetnünk. A továbbiakban röviden egyenként ismertetem az általam felhasznált Azure szolgáltatásokat.

Azure IoT Hub

A szolgáltatás, ahogy a neve is sejteti, kifejezetten IoT témájú problémák kezelésére ad egyszerű, jól működő megoldást. A termék alapvetően az eszközök és a felhő közötti kommunikációt hivatott leegyszerűsíteni. Ez hatalmas előnyt jelent egy hagyományos kliens szerver architektúrával szemben, hiszen nem kell foglalkoznunk külön a kapcsolat kiépítés, fenntartás és menedzsment problémájával. Szintén

megkerülhetjük a terhelés, illetve a terhelés elosztás jelentette kihívásokat, nem beszélve a szerver rendelkezésre állásának megoldásáról és egyéb további buktatókról.

Az IoT Hub tehát teljesen felügyelt szolgáltatást, és megbízható, biztonságos két irányú kommunikációt biztosít, akár több milliós nagyságrendű IoT eszköz között. Ezen felül lehetőségünk van fájlok küldésére, a beépített útválasztó lehetőség révén képesek vagyunk a beérkezett üzeneteket más Azure szolgáltatások felé továbbítani. Az eszközök szolgáltatásba történő regisztrálásukkor úgynevezett „device twin" -ek jönnek létre. Ezek egyéni biztonsági hitelesítő adatok segítségével folytatnak biztonságos kommunikációt. A regisztrált eszközökről JSON-ben tárolt metaadatok segítségével bármikor végezhetünk szinkronizációs vagy egyéb karbantartási feladatokat, mint például szoftver frissítést, anélkül, hogy azt egyenként, eszközönként kellene megoldani. Az eszközök képesek az MQTT v3.1.1, HTTPS 1.1 vagy AMQP 1.0 protokollok használatára a kommunikáció során, amelyek kifejezetten IoT megoldások. A részletesebb szolgáltatás működés bemutatásába terjedelmi okokból kifolyólag, most nem merülök bele [11].

Azure Storage

Az Azure Storage szolgáltatás önmagában egy biztonságos felhőalapú adattárolást valósít meg. A szolgáltatás keretein belül három alszolgáltatás érhető el a Blob Storage, File Storage és Queue Storage, amelyek mindegyike magas rendelkezésre állású, méretezhető és redundáns tárolást tesz lehetővé, miközben a karbantartással kapcsolatos problémákat a Microsoft látja el.

A Blob Storage egy fájl alapú adat tárolást tesz lehetővé, ahol könyvtár szerű struktúrába, mappákba rendezve helyezhetjük el a tárolni kívánt adatokat, azok típusától függetlenül. A tárolóban elhelyezett adatok elérése széles körben támogatott Node.js, Java, .NET nyelvektől kezdve, az URL címekig, a REST szolgáltatásokkal bezárólag. A File Storage előnye, hogy egyazon fájlhoz több virtuális gép is hozzá fér. A Queue Storage üzenetek üzenetsorban történő, feldolgozását teszi lehetővé [12].

Stream Analytics

Az Azure Steam Analitycs egy felügyelt eseményfeldolgozó szolgáltatás, amellyel valós idejű adatfolyamon végezhetünk elemző jellegű számításokat vagy műveleteket. Ezek a beérkező adatok származhatnak egyéb szolgáltatásokból, eszközökből, érzékelőkből vagy akár webhelyekről is. A vizsgált adatokban jellemző

mintázatok alapján, vagy szabályok segítségével szűrhetünk ki nem kívánatos értékeket, vagy aktiválhatunk más folyamatokat, műveleteket, például riasztásokat.

A szolgáltatás egy adatfolyam forrásból kapja az adatokat, például IoT Hub vagy Blob Storage szolgáltatásból, amely adatfolyamon az előre deklarált feladat végrehajtódik és ezután az egyes adatokat az előre megadott kimenetére irányítja. Ehhez egy SQL lekérdezés jellegű nyelvet használ, amely lehetővé teszi a streamelt adatok időtartamon belüli szűrését, rendezését, összesítését. A kimenetként megadott végpont az adatok felhasználását tekintve igen változatos lehet: Visszakerülhetnek egy adatbázisba, üzenetsorba kerülhetnek, vagy továbbíthatjuk őket egy Power BI irányítópultnak [13].

PowerBI

A Power BI egy Microsoft által fejlesztett, üzleti elemzések készítéséhez felhasználható szolgáltatás. Segítségével az elemezni és vizualizálni kívánt adatainkat egyszerű és hatékony módon kezelhetjük egy felületen.

Az adatokból személyre szabható szűrők segítségével választhatjuk ki azok megjeleníteni kívánt részhalmazát. A szolgáltatás továbbá beépített diagramok és grafikonok számtalan lehetőségével és változatával segíti az adatok megfelelő ábrázolását, nem beszélve a diagram áruház nyújtotta lehetőségekről, ahonnan egyéni tervezésű grafikonokat vehetünk igénybe saját felületünk elkészítéséhez. Az elkészített ábrázolási felületek interaktívak, és a kívánt felhasználók között széles körben könnyen megoszthatóak. A felhő alapú megoldás alkalmas önmagában az adathalmazaink tárolására, amelyekből jelentéseket készíthetünk, majd az egyes jelentéseket közzé téve irányítópultokat hozhatunk létre. A Power BI környezete elérhető asztali változatban, mobil alkalmazásként, vagy akár a böngészőből is [14].

Service Bus

Ha kettő vagy több fél (szolgáltatás) üzenetet szeretne cserélni, vagy valamilyen megbízható kommunikációs csatornára van szükségük információ cseréhez, a Service Bus megfelelő megoldásnak bizonyulhat. Hasonlóan egy posta szolgáltatáshoz, az üzenetek kellőképpen rugalmas és megbízható kézbesítését valósítja meg. A különféle esetek kezelésére három különböző kézbesítési típust használ: az egyszerű üzenetsorokat, „publish-subscribe” mechanizmust, illetve közvetlen üzenet átadást.

A felhasználók névtereket hoznak létre, amely névtérben a névtér által meghatározottak alapján zajlik az adott kommunikációs folyamat. Ez történhet egyirányú kommunikációt feltételező üzenetsorokban, ahol az egyes üzenetsorok közvetítő csatornaként működnek és tárolják az üzeneteket, amíg azokat nem fogadják. Minden üzenetet egy fél fogadhat. Ezen kívül lehetőség van topikonkénti, témakörönkénti üzenet közvetítésre, ahol csak az adott témára feliratkozott és a témához tartozó üzeneteket kapja meg a címzett. A közvetlen üzenet átadás két irányú kommunikációt tesz lehetővé. Itt az előző kettő mechanizmussal szemben az üzenetek nem tárolódnak [15].

Logic App

Logic Apps segítségével az Azure nyújtotta felhőszolgáltatásokból automatizált és skálázható munkafolyamatokat hozhatunk létre, a lehető legegyszerűbb módon.

Munkafolyamatok létrehozásával alkalmazások, rendszerek és egyéb logikai végpontok közötti információ és adatcsere viselkedést automatizálhatunk a legkülönfélébb megoldásokkal. Minden általunk elkészített folyamat egy esemény indítóval aktiválódik, amely esemény bekövetkezte után a munkafolyamatban összefoglaltak végrehajtódnak. Ezeket a munkafolyamatokat egy vizuális tervező segítségével hozhatjuk létre az Azure portálján. A munkafolyamatokban létrehozhatunk adatátalakító és folyamatvezérlő műveleteket is, mint például feltételes és választás alapú utasításokat vagy ciklusokat, hurkokat, esetleg elágazásokat. Például egy figyelmeztető üzenet beérkezésekor e-mailes értesítést indíthatunk egy SMTP szolgáltatás bevonásával vagy naplózhatjuk egy riasztásokat kezelő adatbázisba [16].

5 A proof of concept rendszer

A továbbiakban az általam elképzelt szállítmányozást támogató szolgáltatást megvalósító rendszert fogom bemutatni, amelyet annak érdekében készítettem el, hogy bizonyítsam a korábbi koncepcióm és architektúrám működőképességét. Ezt a rendszert az ismertetett architektúra és architektúrális elemek alapján építem fel és számos olyan gyakorlati megfontolással egészítem majd ki, amely egy éles környezetben történő alkalmazás esetén figyelembe kell venni.

A rendszer megalkotása előtt azonban célszerű tisztázni azokat az elvárásokat, amelyeknek eleget téve a rendszer alkalmasnak bizonyul feladatának ellátására.

5.1 A rendszer követelményei

Munkám megkezdése előtt kulcsfontosságú az elkészítendő rendszer tekintetében azon elvárások, kívánalmak megfogalmazása, amelyek teljesülése esetén az elkészített szolgáltatás alkalmasnak bizonyul a rá háruló feladatok, problémák megoldására. Ha a rendszer architektúrájából indulunk ki, célszerű elsősorban a legalsóbb fizikai eszközök jelentette rétegbéli elvárásokat tisztázni.

Rendelkezésreállás, annak biztosítása érdekében, hogy az adatgyűjtő egység feszültségforrásra való csatlakoztatását követően, rögtön elinduljon a program, amely a felhővel történő kommunikációt menedzseli, illetve az adatok gyűjtését végzi. Így nem kell különösebb feladatokat elvégezni az eszköz üzembehelyezéséhez.

Folyamatos szolgáltatás, hogy az eszközön futó programban fellépő bármilyen hiba fellépését követő leállás után, beavatkozás nélkül újra induljon és a megszokottak szerint folytassa futását.

Kapcsolat létesítés: eszközünk futása közben fontos, hogy a lehetőségekhez mérten folyamatos kapcsolatban álljon a felhőszolgáltatással, amely az elküldött üzeneteket dolgozza fel.

Adatgyűjtés: a szenzorokból történő periodikus vagy folytonos adatkinyerés biztosítása. Az adatközpontú szolgáltatásunk felé a legfontosabb és egyben triviális elvárásunk az, hogy képes legyen a csatlakoztatott szenzorokból értelmezhető adatot kinyerni.

Adat mentés: a program futása során gyűjtött adatokat az eszköz képes legyen lokálisan is tömörített fájl formátumban tárolni, így biztosítva az adatok biztonságát, visszakövethetőségét.

Adatküldés: fontos elvárás, hogy a helyileg kigyűjtött adatokat a háttérrendszerrel kialakított kapcsolaton keresztül folyamatosan el tudjuk küldeni feldolgozásra.

Riasztások: elvárható a rendszertől, hogy bizonyos mért adatokról, előre meghatározott szabályok szerint, értesítéseken keresztül, üzenetekben tájékoztassa a háttérrendszert a rendkívüli események bekövetkeztéről.

A rendszerünk háttérfeldolgozásért felelős felhőszolgáltatások felé a következők a követeléseink, elvárásaink:

Rendelkezésreállás, hogy a felhőszolgáltatások minden körülmények között elérhető legyen az eszközök, illetve felhasználók számára, így biztosítva a rendszer akadálytalan működését. Ezért tulajdonképpen a felhőszolgáltatás üzemeltetője a felelős.

Eszköz kommunikáció: a szolgáltatásban résztvevő eszközökkel való folyamatos kapcsolattartás és azok menedzselése. A követelmények kommunikációs vonatkozásai a felhőszolgáltatás karbantartóját terhelik, viszont az eszközökön futó kód és az eszközök menedzselése a mi feladatunk.

Adat mentés, hogy az eszközök által kinyert adatok ne vesszenek el, később is fel lehessen őket használni. Az adatok naplózása és mentése kulcsfontosságú lépés minden további szolgáltatás tekintetében, hiszen ezekből az adatokból bármikor vissza kereshetünk, ha szükség lenne rá.

Adatok ábrázolása, fontos követelmény, hiszen felhasználóink és saját magunk számára is elengedhetetlen az amúgy temérdek szöveges, karakteres információ emberi szemnek is emészthető és informatív megjelenítése. A vizualizáció segítségével az adatok sokasága közötti összefüggéseket sokkal könnyebben láthatjuk át.

Értesítések kezelése, annak érdekében, hogy az eszközeink által generált riasztás üzenetekről az illetékes feleket értesíthessük. A felhasználóinkat valamilyen elérhetőség alapján tudnunk kell értesíteni a szállítás során keletkező rendellenességekről, fontos történésekről.

Terjeszthetőség, alatt azt értem, hogy az általunk előállított információt és adatokat szükség szerint széles körben elérhetővé kell tennünk. Ez lehetséges irányítópult

formájában, nyers adathalmazként, Excel dokumentumként, elektronikus napló .pdf kiterjesztésű változatában, esetleg API formájában, más rendszerek által automatikusan.

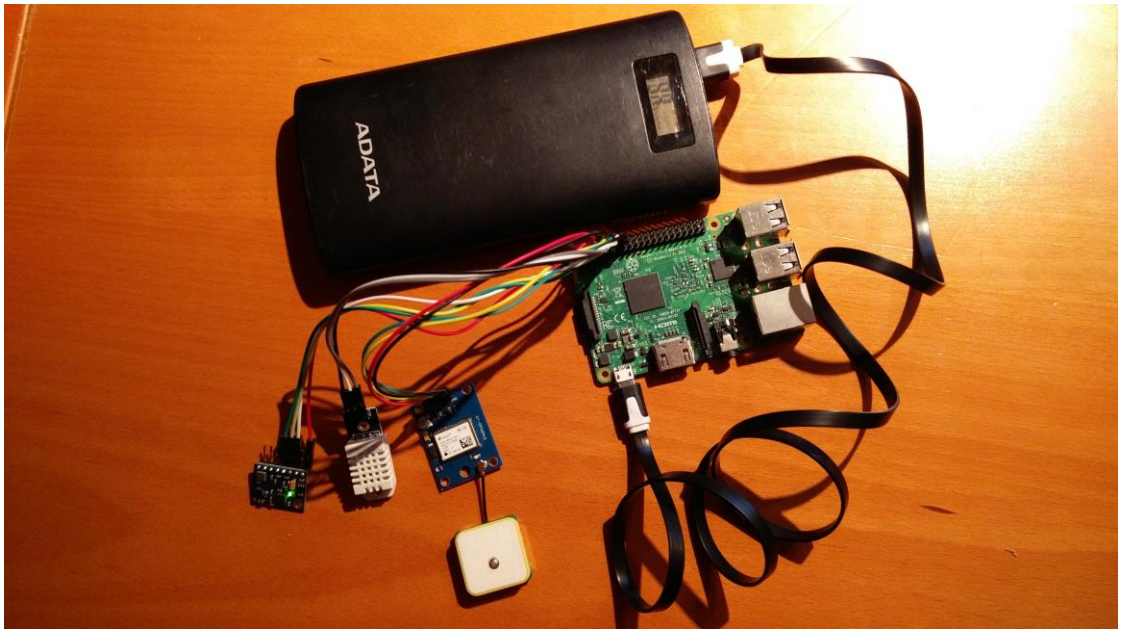
Skálázhatóság, tehát az általunk nyújtott megoldásoknak éppen úgy kell működnie ezres nagyságrendű eszköz és felhasználó esetén is mint, ahogy azt egy esetén tettük meg. Erre az Azure használat alapú fizetési rendszere tökéletes lehetőséget biztosít.

Ezen követelményeknek megfelelve és ezeket szem előtt tartva törekedtem prototípus rendszerem megalkotására, amely részletes leírását a következő fejezetek mutatják be. A továbbiakban a megoldásomban szereplő fizikai adatgyűjtő eszközt telematikai eszköznek fogom hívni, amely egy telekommunikációt alkalmazó adatgyűjtő informatikai eszközt jelent.

5.2 A telematikai eszköz komponensei

Az elkészített rendszerem hardver komponenseit és annak felhasználását ismertetem most részletesen.

A működéséből kifolyólag az eszközöknek és magának a megoldásnak épp annyira kellett kompaktnak lennie, hogy a hordozhatóság ne jelentsen problémát az eszköz folyamatos üzemeltetésében. Végtére az egész eszköz egy kisebb kartondobozban elfért, amelyet bárhova kényelmesen magammal vihettem a működés teszteléséhez. Az 5.2.1-es ábrán látható az elkészített prototípus modell megoldását képező minden hardver komponens és azok összeköttetése.



5.2.1. ábra: Az általam elkészített prototípus modell

5.2.1 A felhasznált Raspberry Pi eszköz és tápellátása

A felhasznált eszköz egy Raspberry Pi 3 Model B volt, melynek legfontosabb tulajdonságait ismertetem.

Az eszköz, lévén egy egykártyás számítógép, nem nagyobb, mint egy bankkártya, ha az alapterületét vesszük figyelembe. Számítási erőforrásait tekintve az adatgyűjtés célját bőven meghaladja, de a fejlesztés és fejlesztőkörnyezetek kényelmes használatát szem előtt tartva már indokoltak az erős hardveres megoldások. Egész pontosan egy ARM Cortex-A53-as típusú, 1.2 GHz órajelű, 64 bites processzor dolgozik az eszközben, amelyet egy 1 GB LPDDR2-es típusú RAM szolgál ki.

A maximális teljesítmény érdekében, a háttértár esetében fontosnak tartottam a lehető legjobb minőségű, legnagyobb olvasási és írási sebességet biztosító microSD kártya használatát. Az általam felhasznált SanDisk Ultra gyártmányú microSDHC kártya 98 MB/s-os olvasási sebességgel és valamivel lassabb írási sebességgel rendelkezett és 16GB tárhelyet biztosított, amely bőven elegendő volt az operációs rendszer és a futtatott programok tárolására. Az eszköz következő fontos csatoló felülete az a 40db GPIO (General Purpose Input Output) pin, amely csatlakozókon keresztül képes az egyes szenzorokkal kommunikálni és egyéb mikroelektronikai eszközöket vezérelni. Ezek közül kitüntetett szereppel rendelkeznek a földelést, illetve tápellátást biztosító, valamint az SPI, I2C, UART és egyéb kommunikációs interfészhez dedikáltan kinevezett pin-ek. A valódi szabad felhasználású pin-ek száma 26, amelyeket a rendszerem megvalósítása során is használtam. A rádiós kommunikációs csatornák közül képes a 2.4GHz-es Wi-Fi (802.11 b/g/n), Bluetooth 4.1 Classic, illetve a Bluetooth Low Energy használatára.

A tápellátás minimális követelménye az 1.2A áramerősségű és 5V-os feszültség forrás, maximálisan pedig 2.5A, illetve 5.1V a megengedett. Ezt egy microUSB csatoló felületen, vagy a GPIO pineken keresztül biztosíthatjuk. A hordozhatóság kérdésében egy külső, hordozható, újratölthető akkumulátor volt a segítségemre. Az Adata gyártmányú Lithium-Ion akkumulátor kapacitása 20000mAh volt, amelyről az eszköz egyszeri töltéssel folyamatos működés mellett 50-55 órát volt képes üzemelni. Az akkumulátor kimenetén 2.1 A erősségű és 5V feszültségű egyenáram folyt, amely tökéletesen megfelelt a Raspberry értékhatárainak.

5.2.2 A felhasznált adatgyűjtő modulok és szenzorok

Az elkészített modellem továbbá három adatgyűjtő szenzort és szenzor modult tartalmazott, amelyekről már korábban szó esett, de itt szeretnék részletes alkalmazásukra is kitérni.

Az eszköz nyomon követhetőségéhez egy GPS modult használtam, amely a föld körül keringő műholdak rádiós jeleiből képes a pozícióját meghatározni. A civil szférában is használatos és elérhető eszközök nagyjából 7-10 méteres pontossággal képesek a pontos földrajzi pozíció meghatározására. Az általam felhasznált Ublox GPS6MV2-es GPS modul egy modell repülőzéshez is használatos eszköz, amely alapvetően jó jelviszonyok között szabad téren működik tökéletesen. Épületeken belül és sűrűn lakott városok magas falai között már kevesebb sikerrel kapunk megbízható jeleket. Mindenesetre a közúti alkalmazásban tökéletesen használható volt, mivel itt semmi olyan objektum nem gátolta a rádióhullámok terjedését, amely a működés rovására ment volna.

A GPS műholdas helymeghatározásra és nyomkövetésre alkalmas eszközünk egy uFL csatlakozón keresztül egy kerámiaantennára csatlakozik a biztos jelforrás érdekében. A modulnak 4 lába van VCC, GND, RX, TX és UART interfészen keresztül kommunikál, amelyhez a Raspberryn dedikált GPIO pin-ek tartoznak. Az egység VCC pin-jét a Raspberry 3.3V-os feszültségű pin-jére kötöttem egy jumper kábel segítségével. Ennek mintájára a GND kivezetést a Raspberry földelésért felelős lábára kötöttem. Az UART kommunikációért felelős RX (Receiver), TX(Transmitter), fogadásra és küldésre kinevezett modul pin-eket a Raspberry erre a célra dedikált GPIO 14 és 15-ös jelzésű lábára kötöttem.

Az olvasáshoz először a Raspberryn engedélyezni kellett a soros felületen történő kommunikációt, amelyet a *„raspi-config”* terminál paranccsal előhozható ablakban lehet beállítani. Ezt követően az egyes GPIO pin-eken be kellett állítani a csatoló felület felhasználási módját ALT0-ra, hogy tetszőleges boudrate-en történhessen ezután a kommunikáció, amelyet a *„pigs 14 4 0”* illetve *„pigs 15 4 0”* paranccsal tehetünk meg. Az adat olvasáshoz 9600-as baudratet használtam, amelyre több leírás ajánlása alapján jutottam.

A következő szenzor, amelyet felhasználtam egy hőmérséklet és relatív levegő páratartalmat mérő szenzor, mivel egy mélyhűtött árut vagy élelmiszert feltételező

rakomány esetén kulcsfontosságú a szállítási hőmérséklet mérése. Ezért fontos volt olyan szenzort választani, amely a valós viszonyokhoz igazodva egy széles skálán képes pontosan hőmérsékletet mérni, akár – 40 fokban is.

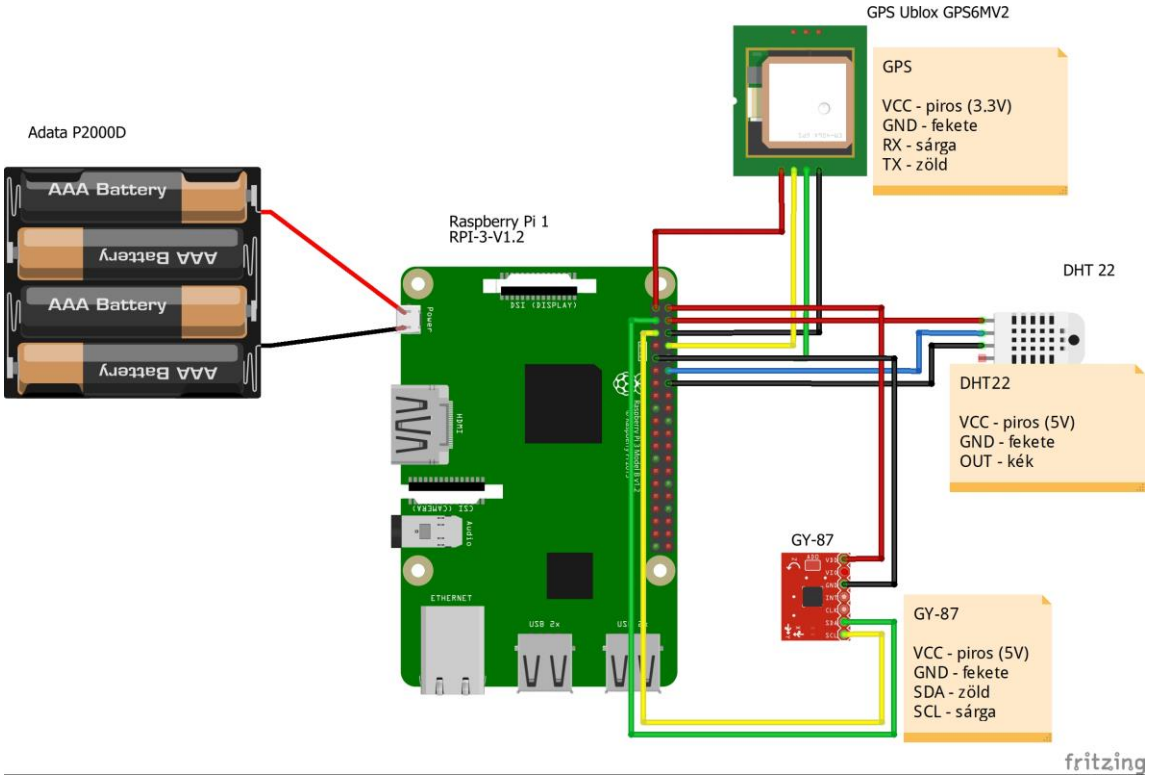
A felhasznált DHT22-es szenzor hőmérséklet mérési tartománya mínusz 40 és plusz 80 Celsius fok közé esik, amelyen belül fél fokos pontossággal mér. A páratartalom mérése esetén értelemszerűen 0-tól 100 százalékig mér az eszköz 2-5 százalékos pontossággal. Az eszköz egyaránt alkalmas 3.3V, illetve 5V-os tápfeszültségről való működtetésre. Használata során 2 másodpercenként olvashatunk friss adatokat az eszköz kimenetéről. Az adatok olvasása érdekében itt nincs szükség különösebb eszköz konfigurációra. Szenzorunknak 3 lába van egy VCC egy GND és egy OUT jelzésű. A szenzor VCC lábát a Raspberry egyik 3.3V-os lábára, a GND lábat a földre csatlakoztatjuk egy jumper kábel segítségével. Az OUT jelzésű lábat a GPIO 18-as lábra kötve a Raspberryn már olvashatjuk is az adatokat a szenzorból.

Az általam használt harmadik szenzor modul három különálló szenzort tartalmaz. A GY-87 névre hallgató, 10 szabadság fokú modul tartalmaz egy MPU6050-es giroszkóp és gyorsulás mérő szenzort egy HMC5883L-es iránytűt, magnetométert és egy BMP180-as nyomás mérő szenzort. A másik két szenzor adatait, az iránytű kivételével, egyaránt felhasználtam a mérések során.

A giroszkóp adataiból fordulás, forgás, illetve borulás vonzatú adatokat tudunk gyűjteni. A gyorsulás mérő a menetirány menti gyorsulás mérése mellett szintén alkalmas a nagy lassulások, fékezések detektálására, illetve a rázkódások és esetleges ütközések következtében fellépő nagy mértékű sebességváltozás mérésére. A nyomás mérő segítségével légnyomás adatokat tudunk elsősorban mérni, amelyből a tengerszint feletti magasság is közelíthető. Egyéb, szintén nyomás mérésre alkalmas szenzorral például tartálykocsik belső folyadék nyomását lehetne figyelemmel kísérni, ilyen szenzort sajnos nem sikerült beszereznem.

A modulnak 8 lába van, amelyek közül az 5V-os VCC_IN jelzésűt, a GND, SCL és SDA kivezetéseket használom fel a kommunikáció során. Mivel a modullal I2C interfészen keresztül lehet kommunikálni, ehhez előbb a Raspberry beállításaiban engedélyezni kell ezt, a már korábban ismertetett konfigurációs parancs segítségével. A VCC_IN lábat egy 5V-os pinre, a GND lábat pedig földre kötöttem. Az SCL lábat a Raspberry GPIO 3-as pin-jére kötöttem, amely dedikált SCL láb I2C kommunikációhoz. Az SDA lábat hasonló módon kötöttem be azzal a különbséggel,

hogy ez a GPIO 2-es pin-jére került. Az adatok olvasása 10 000-es baudraten történt, melynek során egymást követték a modulból kinyert egyes szenzor értékek. Az 5.2.2-es ábrán az elkészített rendszer komponenseinek összeköttetéseit leíró fritzing modell látható.



5.2.2. ábra: Az általam elkészített prototípus kötési modellje

5.2.3 Az eszköz kommunikációs megoldása

Az eszköz által gyűjtött adatokat célszerűen egy rádiós technológiát alkalmazó megoldással lehet eljuttatni a felhő szolgáltatásnak feldolgozásra. A legegyszerűbb megoldást választva, az eszközben található WiFi modult használtam fel erre a célra. A mobil telefonomat hordozható internet csatlakozási ponttá alakítva, képes voltam a már kiépített mobil rádiós hálózaton keresztül a mobil internet csomagomat felhasználva, adatokat küldeni az eszközről. Így mindenhol, ahol a szolgáltató biztosította a mobil internethez való hozzáférést, ott az eszköz képes volt a gyűjtött adatokat a telefonomhoz csatlakozva, a felhőbe küldeni.

5.3 A gyakorlati megfontolások a telematikai eszközben

Szeretném az általam elkészített prototípus eszköz hardveres mgoldásait olyan észrevételekkel és megfontolásokkal kiegészíteni, amelyeket egy valódi, éles környezetben alkalmazható megoldás esetén érdemes szem előtt tartani.

5.3.1 Számítási eszköz és tápellátásának megoldása

Az általam használt Raspberry Pi eszköz bőven meghaladja az adatgyűjtés jelentette feladathoz elvárható kapacitásbéli követelményeket és az ára is igen magas, ha az ésszerű alkalmazásának kereteit vesszük figyelembe. Ennek következtében egy éles környezetben történő alkalmazása pazarló lenne, mind anyagi, számítási és energia felhasználási szempontokból is. Ezen felül a gyakorlatban, a raktér által realizált területről nem elegendő egy helyen adatot gyűjteni. Ennek érdekében az eszköz elhelyezéséből kifolyólag vagy indokolatlan hosszúságú és mennyiségű kábelezést kell felhasználnunk a szenzorok megfelelő pozícióban történő elhelyezéséhez, amely egyéb fizikai akadályokba ütközhet, vagy több adat gyűjtő eszköz együttes helyi rádiós hálózatba kötésével oldjuk meg az adatok kinyerését. Egy ilyen elosztott rendszer alkalmas lehet részletesebb és pontosabb, több fizikai ponton történő adatgyűjtéshez. Ezeket az adatokat aztán egy központi egység összegyűjti és aggregálva küldi tovább azokat a feldolgozó felhőszolgáltatásnak. Méretben, árban, teljesítményben és fogyasztásban is egyaránt alkalmasnak és használhatónak tartom a NodeMCU, illetve WeMos D1 mini Pro platformokat, amelyek mindegyike egy ESP8266-os WiFi képes mikrochipet használ. Ezek az eszközök alvó állapotba kapcsolva, minimális fogyasztással, hosszú ideig képesek működni, adatot gyűjteni.

Szintén a megoldásom prototípus jellegéből kifolyólag egy külső akkumulátorról működtetem a rendszert, amely folyamatos cserélgetése és töltése egy üzemszerűen működő rendszerben elképzelhetetlen. Ehelyett a szállító jármű elektromos hálózatára lenne célszerű rákötni az eszközt, esetleg ennek az energiának a tárolásához egy kisebb a célra dedikált akkumulátort beszerelni. A külön akkumulátor beépítésével a sokat parkoló járművek esetében is megoldható maradna az eszközök folyamatos tápellátása és nem kell attól tartanunk, hogy leáll az eszközünk vagy lemeríti a szállító egység saját akkumulátorját.

5.3.2 Szenzor modulok megoldása

Láthattuk, hogy az általam elkészített megoldás három különböző szenzort vagy szenzormodult is alkalmazott. Ez egy gyakorlati megoldásban nagyon sok problémát vet fel mind hardver mind szoftver oldalon. Nem mindegy például, hogy hány eszközt kell összekapcsolnunk és az sem mindegy, hogy ezeket a kódban hány különböző interfészen keresztül tudjuk elérni.

Ezért előre mutatónak találom szállítmány specifikus szenzorcsomagok tervezését. Olyan szenzormodulokra gondolok, amely lehetőségek szerint egy nyomtatott áramköri lapon tartalmazzák mindazokat a szenzorokat, amelyek egy bizonyos szállítmányra korlátozódó mérést feltételeznek. Így sok hibázási lehetőséget megspórolva a fizikai összeköttetésekben és a szoftveres oldalon is egy egységes interfészt nyújtva. Például egy fagyasztott árukat szállító rendszerhez külön szenzor csomag tartozna úgy, ahogyan egy élő állatot, vagy veszélyes vegyi anyagot feltételező szállítmányhoz. Természetesen, ehhez mélyrehatóan tanulmányozni kell a szállítmányok kategóriáit és mindegyikhez külön szenzorcsomagot kell terveztetni.

5.3.3 Kommunikációs megoldások

A prototípusom működése során a telefonomról megosztott mobil internet hálózatot használtam az adatok továbbítására. Ez a megoldás szintén alkalmatlan lenne egy valós környezetben történő alkalmazásra, ezért szükségesnek tartom, hogy megemlítsek olyan megoldásokat, amelyekkel megoldható az eszköz kommunikációja a felhő réteggel. Az általam használt megoldáshoz legközelebb álló ötlet egy GSM modul alkalmazása. Egyszerűen a GPS modulba integrálva is lehet kapni GSM modulokat, de külön modulként is beszerezhetőek, amelyek segítségével a kiépített rádiós hálózaton megoldható a kommunikáció.

Ezen felül már számos IoT megoldásokat támogató kommunikációs megoldás létezik a piacon, amelyek használata szintén kézenfekvő és praktikus lehet. Ezek közül kettőt szeretnék megemlíteni, amelyek mindegyike kiépített rádió hálózatot használ. Szerencsére mindkét hálózat erősen elterjedőben van Európa szerte így hazánkban is több helyen alkalmazhatóak már. A LoRa nevezetű hálózat használatához egy külső modult beszerelve akár 10km-es távolságban lévő állomásokkal is kommunikálhatunk 10Kb/s adat sebességgel, amely a telematikai eszközünknek minden szempontból alkalmas lehet. A NarrowBand – IoT megoldás szintén egy külön modul beépítése

mellett teszi lehetővé a kommunikációt, akár 15km-re lévő bázisállomásokkal 100Kb/s adat sebességgel. Említésre méltó lehet még a Sigfox nevezetű kommunikációs megoldás, amely szintén nagy hatótávolságú rádiós adat továbbitást tesz lehetővé. Ennek a szállítmányozásban történő alkalmazásának egyetlen akadálya lehet, mégpedig a hálózati sávszélessége, amely 100 bit/sec adatküldést tesz lehetővé.

A továbbiakban szeretnék kitérni az általam elkészített telematikai eszközön futó kódra, amely az adatgyűjtést és az ezzel kapcsolatos feladatokat végzi.

5.4 A telematikai eszközön futó kód bemutatása

A továbbiakban azokat a szoftveres megoldásokat fogom ismertetni, amelyeket az elkészített rendszerem működtetése során használtam fel. Röviden kitérek a környezetre, amely lehetővé teszi a kód futását, ezt követően pedig bemutatom magát az általam írt adatgyűjtésért és továbbításért felelős szolgáltatást.

5.4.1 A kód futtató környezetének bemutatása

Ahogy azt korábban már említettem a Raspberry Pi-n, egy külön az eszközre optimalizált Debian operációs rendszer fut, amely a Raspbian névre hallgat. Az operációs rendszer lemezképfájlja ingyenesen elérhető az interneten. Ezt egy Etcher nevű program segítségével másoltam fel a már korábban ismertetett SD kártyára, amelyet így egy bootolható meghajtóként ismert fel az eszköz az indulása során. Az operációs rendszer telepítését követően alapvető biztonsági és konfigurációs lépéseket tettem az eszköz védelmének érdekében. A munkám megkönnyítése végett egy VNC (Virtual Network Computing) szolgáltatáson keresztül használtam, értem el az eszközt, így a fejlesztés a laptopomon keresztül is történhetett.

Az általam elkészített szolgáltatást JavaScript nyelven írtam meg, amely futtatása Node.js környezettel történt. A Node.js egy nyílt forráskódú JavaScript kód futtatására alkalmas független szoftverrendszer, esemény alapú, aszinkron I/O műveletekkel, amelyet skálázható webszerverek készítésére hoztak létre. A Node.js 6.13.1-es változatát telepítettem fel az eszközömre és a környezethez tartozó npm csomag menedzser segítségével telepítettem a további, általam felhasznált szoftverkönyvtárakat.

A JavaScript kód írásához a Visual Studio Code nevű fejlesztőkörnyezetet telepítettem fel, amelyet több helyen is ajánlottak a Raspberryn történő

szoftverfejlesztéshez. Ezt az ingyenes, nyílt forráskódú környezetet a Microsoft első sorban Linux és OS X operációs rendszereken történő fejlesztéshez hozta létre. Továbbá az általam írt Node.js szolgáltatások karbantartásához egy PM2 nevű folyamat menedzser programot telepítettem. Ennek segítségével boot folyamat utáni indítást tudtam ütemezni, tehát az eszköz indulása után rögtön elindult az általam írt szolgáltatás is. Ezen felül a PM2 szolgáltatás azonnal újra indítja a gondjaira bízott kódot, ha véletlenül leállna az.

5.4.2 Az adatgyűjtés menete és a gyűjtött adatok értelmezése

Ebben az alfejezetben az általam írt adatgyűjtéshez felhasznált kódokat fogom ismertetni. Minden szenzorhoz külön kódrészlet tartozik egy szenzornév.js kiterjesztésű fájlban, amely az adatok olvasásáért és gyűjtéséért felelős. Ezeket a kódrészleteket egy kliens szolgáltatást megvalósító kódból érem el, amely adott időközönként lekérdezi a szenzorokhoz tartozó kód segítségével a szenzorokon mért pillanatnyi értékeket.

DHT22.js

Kezdve a szenzoros adatolvasást megvalósító kódokkal, azon belül is a DHT22 -es hőmérséklet és relatív páratartalom mérő szenzorhoz tartozó kóddal:

A szenzort egy hozzá tartozó könyvtár segítségével érem el, amelyet az npm csomag menedzser segítségével telepítettem fel. A kódban megjelenő, az eszközt reprezentáló objektumunknak a példányosítása során meg kell adnunk, hogy a Raspberry melyik GPIO pinjén keresztül szeretnénk az adatokat olvasni. Természetesen onnan, ahova előzőleg kötöttük, tehát a GPIO 18-as lábáról. Az adatok olvasása a példányosított eszközön keresztül történik a rajta keresztül hívható read() metódus meghívásával. Ekkor a kódunk az adott pillanatban leolvassa a GPIO pin-en található értékeket és azt Celsius fokban mért hőmérsékletté, illetve százalékos formában megadott relatív páratartalom értékké alakítja. A read() metódussal kiolvasott objektum humidity része a páratartalmat, a temperature része pedig a hőmérséklet értéket adja vissza. Az általam készített DHT22.js kód „exports” kulcsszóval ellátott metódusain keresztül visszakapjuk a pillanatnyi hőmérséklet és páratartalom értékeket egy külső kódból hívva. Például a kliens szolgáltatás kódjában meghívva a readHumidity vagy readTemperature metódusokat megkapjuk az aktuális mért értékeket.

GY-87.js

A GY-87-es szenzor modulból 2 szenzor adatait használom fel, a BMP180-as nyomásmérőét, illetve az MPU6050-es giroszkóp és gyorsulás mérőét. Mindkét szenzor esetében a Johnny-Five nyílt forráskódú JavaScript kód könyvtárat alkalmazom a szenzor adatok olvasásához. A könyvtár részletes API leírást nyújtva próbál, minél szélesebb körben megoldást adni különböző IoT megoldásokban használatos eszközökkel történő szenzoros adat olvasásra és interakcióra. Sok példakód, és kötési rajz található a dokumentációk között, amelynek nagy hasznát vettem.

A BMP180-as szenzorból történő adat olvasáshoz egy GY87.js nevű fájlban foglaltam össze a később meghívandó kódot. Az adatok olvasásához a későbbiekben itt is elég a readPressure, illetve readAltitude metódusok hívása, amelyek visszaadják a szenzorból kiolvasott nyomás és földfelszín feletti magasság értékeket. Ezeket a metódusokat szintén az „exports" kulcsszóval tesszük külső kódból való hívásra alkalmassá ahogy azt a DHT22-es kódjában is láthattuk.

A GY87.js kód működése során először példányosítunk egy Raspberry objektumot és egy BMP180-as szenzor objektumot az API-nak megfelelően, majd egy ciklusban folyamatosan olvassuk a szenzorból elérhető értékeket. Minden ciklusban a megfelelő értékeket egy változónak adjuk át, amelyet külső kódból a korábban megnevezett metódusokon keresztül olvashatunk ki. A kinyert nyomás értéket Pa-ban kapjuk meg, a tengerszint feletti magasság értéket pedig méterben.

Az MPU6050-es szenzor esetében szintén a már említett Johnny-Five könyvtárat használom fel. Az adatolvasást realizáló kód itt is a GY87.js fájlban került implementálásra. Ebből a szenzorból 20 különböző jelentéssel bíró értéket olvasok ki, ezért nem részletezem az ezek olvasásához használt metódusok neveit. Ezek az adatok X, Y és Z tengely menti gyorsulást, fordulást és egyéb ezekből származtatható értékek olvasását teszik lehetővé külső kódból, szintén az „exports" kulcsszó használatával.

A kód inicializálása során az API-nak megfelelően itt is szükséges egy Raspberry objektum, illetve egy MPU6050-es szenzornak megfelelő objektum példányosítása. A szenzorból minden futási ciklusban az alábbi értékeket olvasom ki és írom egy lokális változóba. Gyorsulás értékek az X, Y, Z tengelyen, illetve az ezekből származtatható eredő irány menti gyorsulás G-ben mérve (1G = 9,82m/s^2). Ezen felül mérem két adat olvasás közötti, két mért gyorsulás érték közötti maximális eltérést.

Ennek azért van kitüntetett szerepe, hogy egy 12 másodperces mintavételezés mellett se késsünk le a legnagyobb gyorsulás érték változásokról. A 12 másodperces mintavételezésre az igénybe vett ingyenes szolgáltatás korlátai miatt volt szükség, amely napi 8000 üzenet küldését tette lehetővé. Így adódott, hogy az eszköz 12 másodpercenként küldött üzenetet annak érdekében, hogy folyamatos működés mellett is bele férjen a napi keretbe.

A giroszkópból X, Y, Z tengelyen történő elfordulás értékeket olvashatunk radiánban, illetve az elfordulás sebességét mérhetjük fok/másodpercben. Ezen felül az eszköz orientációjáról tanúskodó egyéb, a korábban említettekből származtatott adatokat is olvashatunk. Az adatokat mindkét szenzor esetén (BMP180, MPU6050) 10 000-es baudraten olvasom ki, amely értéket a Raspberry Pi „/boot/config.txt”-ben adok meg a *„dtparam=i2c_arm_baudrate=10000”* sorral.

GPS.js

A GPS szenzorból történő adat olvasásért felelős kódot a GPS.js fájlban implementáltam. Itt valósítom meg a GPS modullal befogott rádiós jelekből kinyerhető adatok olvasását egy ciklusban. A kinyert adatok helyi változókba mentve, egy a már korábban ismertetett olvasó metódusok segítségével érhetők el a kódon kívülről. A GPS modulból kinyert adatok feldolgozásához egy GPS.JS nevű könyvtárat használok fel, amely lényegében egy karakterlánc feldolgozást végez a neki átadott értékeken. A kimenetet egy JSON objektumban kapjuk vissza, amelyből már könnyen hozzáférhetünk a kívánt GPS adatokhoz.

A kódban egy soros porton keresztül olvassuk a *„/dev/ttyS0”* fájlból az UART kommunikáció során érkező GPS adatokat 9600-as baudraten. A beérkező karakterláncot egy parser segítségével továbbítjuk egy, a GPS könyvtárunkból példányosított karakterlánc feldolgozónak. Az adatok soros portról történő olvasása és az olvasott értékek feldolgozása folyamatosan, megszakítás nélkül történik. A ciklus végén a feldolgozott, már JSON formátumban reprezentált különféle GPS adatokat helyi változóknak adjuk át, amelyeket a külső kódból érkező kérések zavartalanul olvashatnak.

A teljesség igénye nélkül, az általam legfontosabbnak tartott és feldolgozott adatok az alábbiak: szélességi és hosszúsági koordináták, tengerszint feletti magasság,

pillanatnyi sebesség, pontos idő és ezen adatok pontosságát leíró egyéb mért adatok. A tengerszint feletti magasság méterben, a pillanatnyi sebesség km/h-ban olvasható.

RaspAlfaClient.js

A telematikai eszköz felhővel történő kommunikációja a RaspAlfaClient.js névre hallgató fájlban került implementálásra. Ez a kód lényegében az előbb ismertetett három szenzor adatot olvasó kód read() metódusainak ciklusban történő hívásával begyűjti az egyes szenzorokból a kívánt adatokat, majd tovább küldi őket egy üzenet formájában a felhőnek. Ehhez természetesen implementálni kell az Azure IoT Hubbal való kapcsolat kiépítését. A kapcsolat kiépítése egy callback függvénnyel történik, amely egy setInterval metóduson keresztül 12 másodpercenként lekérdezi a korábban megírt kódjaink read() metódusain keresztül a szenzorokon aktuálisan mért értékeket. Ezekből az értékekből aztán egy karakterláncot készítünk, amelyből egy JSON objektumot csinálunk. A felhőnek már ezt az objektumot küldjük tovább.

5.4.3 Az adatok küldése, tárolása és riasztások kezelése

Az egyes elküldeni kívánt adatok esetét az imént már részleteztem. A JSON-be konvertált adatainkból egy Message típusú objektumot hozunk létre. Ezt a Message objektumot küldjük el az IoT Hub-nak feldolgozás céljából. Ezen felül a RaspAlfaClient.js kódban egyéb más folyamatok is lejátszódnak, amelyekre szeretnék most kitérni.

Például az üzenetek küldése mellett minden setInterval függvény hívás után az aktuálisan kiolvasott szenzor adatokat egy lokális szöveges fájlhoz fűzöm. Ezeket a fájlokat aztán napi rendszerességgel egy zip fájlba tömörítem, és eltárolom őket az eszköz háttértárán. Ez naponta 440kB tárhelyet igényel, amely igazán nem sok. Így az eszközön helyileg megvalósított adatnaplózás egy biztonsági lépést jelent az adatok tárolásában és visszakereshetőségében. A már korábban említett egyszerű Message objektumban történő adatküldésen felül külön lehetőségünk van riasztás jellegű tartalom csatolására. Ezeket a riasztásokat saját döntésünk szerint, egyéni meglátás alapján állíthatjuk be a mért adat értékek alpján. A riasztásokat a Message objektum properties mezőjéhez kapcsolva juttathatjuk el a felhő végpontba. Például, ha szeretnénk riasztást küldeni, hogy ha a hőmérséklet meghaladta a 27 Celsius fokot azt így tehetjük meg:

”*message.properties.add('HighTemperature',(temperatureDHT22>27)?'true':'false');”*.

5.5 A telematikai eszköz gyakorlati szoftveres megoldásai

A gyakorlati megfontolások a telematikai eszközön futó kód teljes megváltoztatásával járhatnak, amint az eszközt lecseréljük. Eszközcsere esetén nem garantálja semmi azt, hogy az általunk felhasznált kis fogyasztású eszköz képes lesz a kódunk futtatására. A gyakorlatban az adott eszközre szabottan érdemes az adatgyűjtő szolgáltatást megírni, célszerűen egy hatékony és kellőképp alacsony szintű nyelven.

Erre azért van szükség, hogy a lehető legkisebb energiafelhasználás és számítási kapacitás felhasználásval történjen meg az adatok gyűjtése, vagy az egyéb beavatkozással járó műveletek. Például egy python nyelven megírt szolgáltatás kellőképp alacsony szintű és hatékony lehet ahhoz, hogy egy olyan korlátozott erőforrású eszközön is elfusson, mint az ESP8266. Továbbá az IoT Hubbal történő kapcsolat kiépítés és kapcsolat tartás már python nyelven is támogatott és megoldott.

Szenzorok terén a hardveres megoldások során már megemlítettem a szállítmány specifikus szenzorcsomagok tervezését. Ezekhez a szállítmányonként eltérő felépítésű és tartalmú szenzormodulokhoz célszerű saját kód könyvtárakat készíteni a könnyebb és kényelmesebb adatkezelés érdekében. Ezzel egy egységes és hatékony írási és olvasási interfész nyújtható a szenzormodulokból történő adatok eléréséhez és esetleges aktuátorok vezérléséhez.

A korábban megemlített egyéb rádió technológiás megoldások alkalmazása, mint a LoRa vagy NB-IoT, további szoftveres problémákat és megoldásokat vet fel. Ezen protokollok alkalmazásához szintén eszköz specifikus szoftveres ismeret szükséges a kommunikációs kapcsolatok megfelelő kiépítéséhez és használatához.

Továbbá előre mutató megoldásnak találom a szállító egység CAN-bus-ának összekapcsolását az általunk alkalmazott telematikai eszközzel. Így az általunk gyűjtött és felhasznált adatokat egyéb a CAN-bus-ról származó további pontos adatokkal és mérésekkel egészíthetjük ki.

5.5.1 Cloud to device üzenetek

Az IoT Hub használatával lehetőségünk van a felhőszolgáltatáson keresztül eszközeinknek üzeneteket küldeni. Ezzel a megoldással a prototípus alkalmazása során nem sokat foglalkoztam, de rengeteg gyakorlati alkalmazási és felhasználási ötletet vet fel, amelyekből szeretnék párat megemlíteni.

Tegyük fel, hogy egy szoftver frissítést szeretnénk az eszközeinken futó kódon véghez vinni. Ez az általános használatot feltételezve egy hosszas folyamatot jelentene. Az eszközeinket egyenként elérve kellene külön-külön megoldanunk a szoftverek javítását. Ehelyett egy, az eszközön futó jól megírt szoftver képes a felhőből érkező üzenetek alapján cselekedni. Így például, ha utasításként kapja, hogy frissítenie kell az aktuális adat gyűjtő szoftvert, akkor a felhőből kapott friss szoftveres megoldásra automatikusan lecserélheti az eddigi adat gyűjtő kódot anélkül, hogy egyenként kellene ezt nekünk megtennünk. Ily módon a teljes eszközpark szoftver frissítése, akár néhány gombnyomással megoldhatóvá válik. A felhőből az eszköznek címzett üzenetek további felhasználási módját is el tudom képzelni. Például, távolról tetszés szerint paraméterezhető és így testre szabhatóbb riasztások kialakításánál is előre mutató lehet ezeknek az üzeneteknek az alkalmazása.

Tegyük fel, hogy egy mélyhűtött árut szállító egységnél be van állítva, hogy riasztást küldjön, amint a hőmérséklet -10 fok fölé emelkedik. Ugyan ennek a szállító egységnek pár órán belül már egy másik szállítmányt kell kezelnie, amelynél viszont csak 0 fok feletti hőmérsékletről szeretnénk értesítést kapni. Ilyen és ehhez hasonló esetekben célszerű a riasztások generálásának távolról történő testreszabhatóságának megvalósítása.

A későbbiekben akár egy felhasználói felületen keresztül a szállíttatást megrendelő fél beállíthatja, hogy milyen esetekről szeretne értesítést kapni. Ezt a kérést egy közbülső szolgáltatás feldolgozza és elküldi a megfelelő szállításban résztvevő eszköznek alkalmazásra. Ezután az eszköz már csak a kapott instrukciók alapján jelez, riaszt az értesítendő fél számára.

5.6 A felhő réteg szoftveres megoldásai

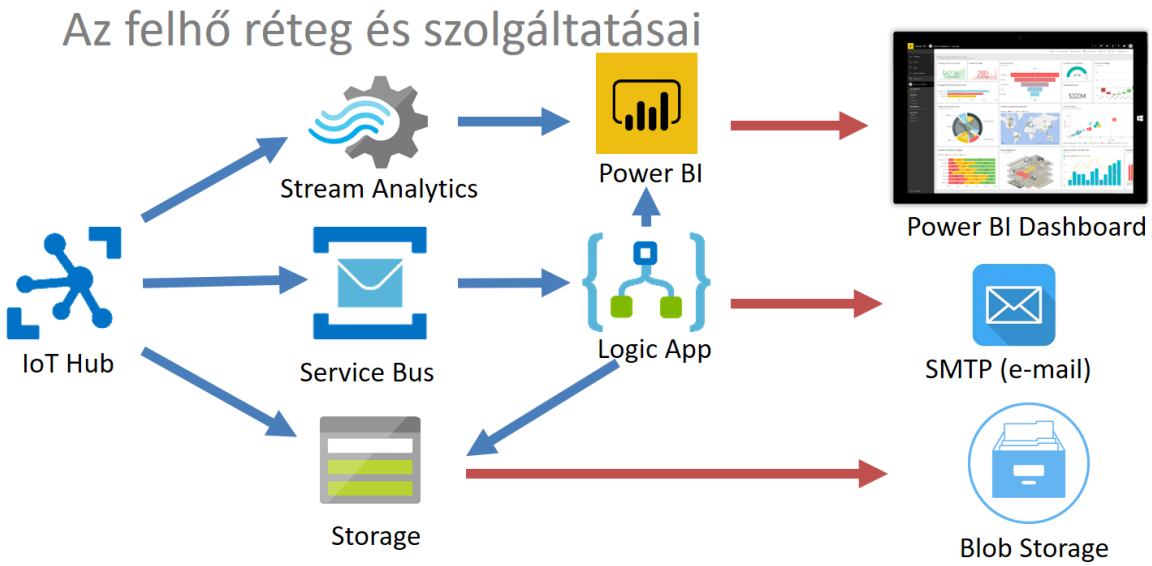
Ebben a fejezetben szeretném részletesen bemutatni az általam felhasznált felhőszolgáltatások működését és azt, hogy ezek segítségével, hogyan valósítottam meg az általam elképzelt okos szállítmányozás nyújtotta szolgáltatásokat magát a háttérrendszert. Háttérrendszer alatt azokat a szoftvereket és szoftveres megoldásokat értem, amelyek valamilyen módon az adatfeldolgozás, illetve adat megjelenítés és tárolás folyamatában részt vesznek.

5.6.1 Azure felhőszolgáltatások

Az általam felhasznált felhőszolgáltatások mindegyike elérhető a Microsoft Azure szolgáltatásai közül. Ehhez csak egy felhasználó fiókot kell regisztrálnunk, amelyen keresztül hozzáférést kapunk az Azure Portal nevű webes alkalmazás felülethez. Ez megtörténhet egy ingyenes fiók keretében, amelyhez egy 170 eurós szabadon felhasználható keretet kapunk a szolgáltatásokra, vagy egy fizetős fiókon keresztül, ahol a felhasznált szolgáltatásokért magunknak kell fizetnünk. Sajnos az ingyenes fiók csak egyszer vehető igénybe, amely már megtörtént egyszer az esetemben, így a fizetős megoldást kellett választanom.

Az irányítópultnak nevezett online webes felületen keresztül, erőforrásnak nevezett csoportokba sorolva történik a szolgáltatások igénybevétele és számlázása. Egy erőforrás csoportot létrehozva, az Azure Marketplace-ről tetszőleges mennyiségű és fajtájú szolgáltatást vehetünk igénybe a létrehozott erőforráshoz csatolva.

A továbbiakban egyenként bemutatásra kerülő szolgáltatások egymással való kapcsolatát írja le az 5.6.1-es ábra, amely egyben a gyűjtött adatok áramlásának útvonalát is bemutatja a szolgáltatás végpontokig. A kék nyíllal jelölt irányított utak az adatok feldolgozásában résztvevő szolgáltatásokat kötik össze. A piros nyilak pedig a szolgáltatások kimeneteként nyújtott már feldolgozott adatokból előállított információ útját jelzik.



5.6.1. ábra: A felhőszolgáltatások közötti adat utak leírása

IoT Hub

Az általam elsőként kipróbált és kulcsfontosságú szolgáltatás az IoT Hub. A szolgáltatás létrehozása során az elvárásainknak megfelelő módon skálázva választhatunk felhasználói csomagot. Az általam választott ingyenes csomag napi 8000 fél kB-os üzenet csomag küldését teszi lehetővé. Ez pont elegendő a szolgáltatás egy eszközzel való kipróbálásához. Az eszközöm folyamatos működését feltételezve és a napi 8000 üzenetes keretből kiindulva 12 másodpercenként küldtem üzeneteket az eszközömről, hogy biztos ne lépjem túl a napi keretet. Ha egy üzenet meghaladta a fél kB-os keretet, akkor az üzenet minden fél kB-ja után felszámolt egy elhasznált üzenet egységet a szolgáltatás.

A szolgáltatás használatának első lépése az IoT eszközök felvétele. Ezt többféle módon is megtehetjük. Talán a legegyszerűbb módja az általam is végig járt: A telematikai eszközön egy konzol alkalmazást írtam, amely futása során létrehoz egy eszköz identitásnak nevezett azonosítót az általam létrehozott IoT Hub szolgáltatáshoz. Ehhez mindössze a már korábban létrehozott IoT Hub-om *„connection string”* nevű azonosítójára volt szükségem, hogy egyértelmű legyen az eszköz és a Hub közötti kapcsolat. Az eszköz regisztrációs folyamata során kapott kulcs segítségével autentikálja magát a későbbi Hub-ba történő üzenetküldések során.

A szolgáltatáson keresztül regisztrált eszközeinkről egy úgy nevezett Device Twin formájában metaadatokat kezelünk, mint például, egyedi azonosító, mikor volt utoljára aktív, mikor küldött üzenetet, elérhető-e, vagy milyen verziójú kódunk fut rajta. A Device Twin-eken eszközölt műveletek formájában, például egy szoftver frissítést ütemezve az „alfa” jelzéssel ellátott Twin-eken, lényegében a fizikai eszközeinken ütemezünk karbantartási munkálatokat. A Device Twin tulajdonképpen az eszközünk virtuális leképezése. Az eszközökből a Hub-ba érkező üzenetek feldolgozására és monitorozására számos lehetőségünk van. Az iothub-explorer nevű Hub szolgáltatással például monitorozhatjuk a Hubnak érkező eszköz üzeneteket. A monitorozáson túl a Hub-ból saját kézzel írt alkalmazások segítségével olvashatjuk a beérkező üzeneteket.

A beérkező üzeneteket végpontok felvételével, más Azure szolgáltatásoknak közvetíthetjük. A beépített végpontokon túl saját végpontokat felvéve például Azure Storage tárolókba, Service Bus üzenet sorokba, vagy Event Hub szolgáltatásnak adhatjuk át az üzeneteinket. Sajnos az ingyenes megoldás csak egy szabad választású végpont felvételét engedélyezi. Így egymás után több megoldást is kipróbálva először

egy Azure Storage-be irányítottam a beérkező üzeneteket. Ezt később lecseréltem egy Service Bus üzenetsorra, amely már a végső megoldásom részét képezte. Az üzenetsorba érkező üzenetekre különböző szabályokat definiálhatunk, mint például mennyi idő után törlődjenek, vagy egyszerre hány üzenet kerülhessen be a sorba. Még mielőtt az üzenetek egy adott végponthoz eljutnának, szükségünk van útvonalnak nevezett csatornák létrehozására, amelyeken keresztül megtörténik az üzenetek végpontba juttatása.

Egy útvonal felvétele során meg kell adnunk, hogy milyen típusú adatot szeretnénk az útvonalon utaztatni, például eszköz üzenetet, eszköz életciklus eseményt, vagy Device Twin eseményt, valamint azt, hogy melyik végpontnak címezzük ezt. Itt lehetőségünk van egy Query String feltétel segítségével csak azokat az üzeneteket tovább küldeni az útvonalon, amelyekre a feltétel igaznak értékelődik ki.

Az általam elkészített megoldásokban a telematikai eszközömből érkező összes üzenetet továbbítottam először egy Azure Storage tárolóba, majd később egy Service Bus üzenetsorba. Ahogy már említettem a végső megoldásban a Service Bus üzenetsort alkalmaztam. Ezen megoldások segítségével az IoT Hub megoldja az eszközökkel járó kommunikációs és karbantartási kérdéseket, valamint az üzenetek egyéb végpontoknak történő továbbítását egy közbeszúrt adatszűréssel.

Service Bus

Az imént már említett IoT Hub-ban végpontként felvett Service Bus szolgáltatásról és annak személyre szabási lehetőségeiről szeretnék most röviden írni. Szintén az Azure Marketplace-ből kiválasztva a már létező erőforrás csoportjainkhoz kapcsolható a Service Bus szolgáltatás. Értelem szerint a már korábban létrehozott IoT Hub-ot is tartalmazó erőforrás csoportba vettem fel ezt a szolgáltatást is, hogy az egymással történő összekapcsolásuk minél egyszerűbb legyen.

A szolgáltatás létrehozásakor itt is több árazási csoport közül választhatunk attól függően, hogy milyen szinten szeretnénk használni az adott szolgáltatást. Az általam választott minimális költségű profil az üzenetsorokon keresztül történő kommunikációt támogatja és maximálisan 256kB méretű üzeneteket engedélyez, amelyet az én fél kB-os üzeneteim meg se közelítenek. A magasabb költségű profilok lehetőséget nyújtanak Topic-okban történő üzenet kezelésre vagy nagyobb méretű üzenetek továbbítására, de erre csak később térek ki. A létrehozott Service Bus szolgáltatáson belül, az entitások

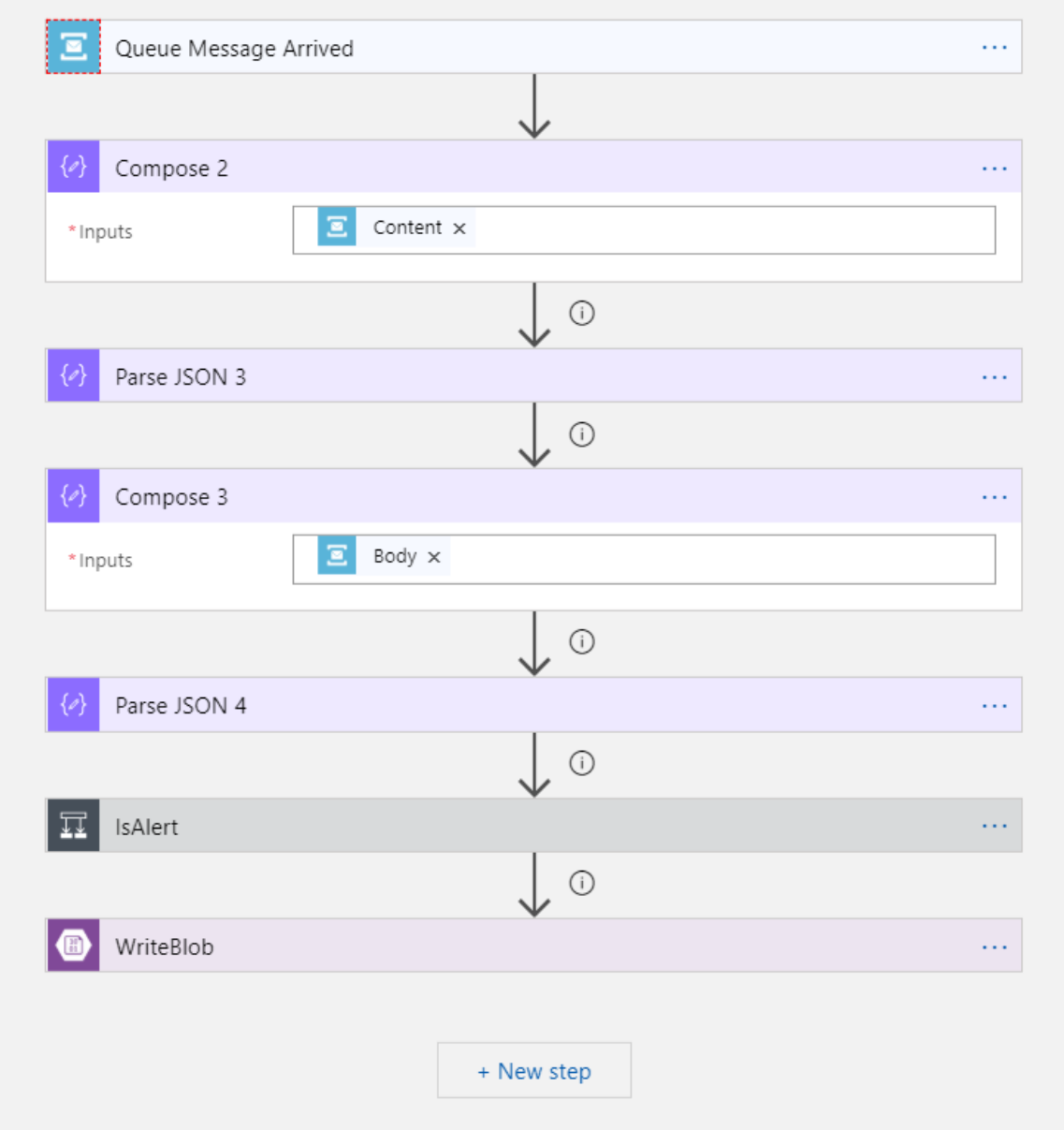
fül alatt lehetőségünk van üzenetsorok készítésére, akár többre is igény szerint. A létrehozandó üzenetsor elnevezése után meg kell adnunk tipikusan GB-ban mérve, hogy mekkora kapacitáson tárolja az üzenetsorba érkező üzeneteket és hogy a beérkező üzeneteket mekkora időtartam után dobja el. Az általam létre hozott „fszbus” névre hallgató üzenetsor 16GB tárhellyel rendelkezik, és a fel nem dolgozott üzeneteket egy napig tárolja majd eldobja az egyszerűség kedvéért.

A Service Bus egyéb monitorozási felülettel is szolgál, amelyen keresztül a létrehozott entitásainkra jellemző diagnosztikai adatokat és mérési metrikákat kísérhetjük figyelemmel. Az egyes üzenetsorokhoz külön kezelői felület is tartozik, amelyen keresztül figyelemmel kísérhetjük az aktuálisan sorban álló üzenetek számával kapcsolatos adatokat és a tárhelyek foglaltságát. Az üzenet sorok FIFO jellegű üzenet tárolást és továbbítást valósítanak meg. Így értelemszerűen, ha az IoT Hub áll a sor bemenetén, a kimenetén is kell állnia egy másik szolgáltatásnak.

Lopic App

A Logic App szolgáltatás tökéletesen alkalmas a Service Bus üzenet sorból való üzenetek kivételére. A Logic App, ahogy azt már korábban is említettem, az Azure nyújtotta szolgáltatásokat összefogva, képes forgatókönyvek formájában esemény alapú műveleti gráfot építeni. A Logic App létrehozásakor a korábban már használt erőforrás csoportunkhoz kapcsolva vesszük fel ezt a szolgáltatást is, ahogy azt eddig is tettük. Így a Logic Appon belül elérhetjük a már korábban létrehozott üzenet sorainkat. A szolgáltatás vezérlőpultján találhatunk egy fejlesztő eszközöket tartalmazó részleget. Az itt található Logic App Designer segítségével hozhatjuk létre azokat a forgatókönyveket, amelyek lépéseit bizonyos események bekövetkeztekor szeretnénk, hogy egymás után végrehajtódjanak.

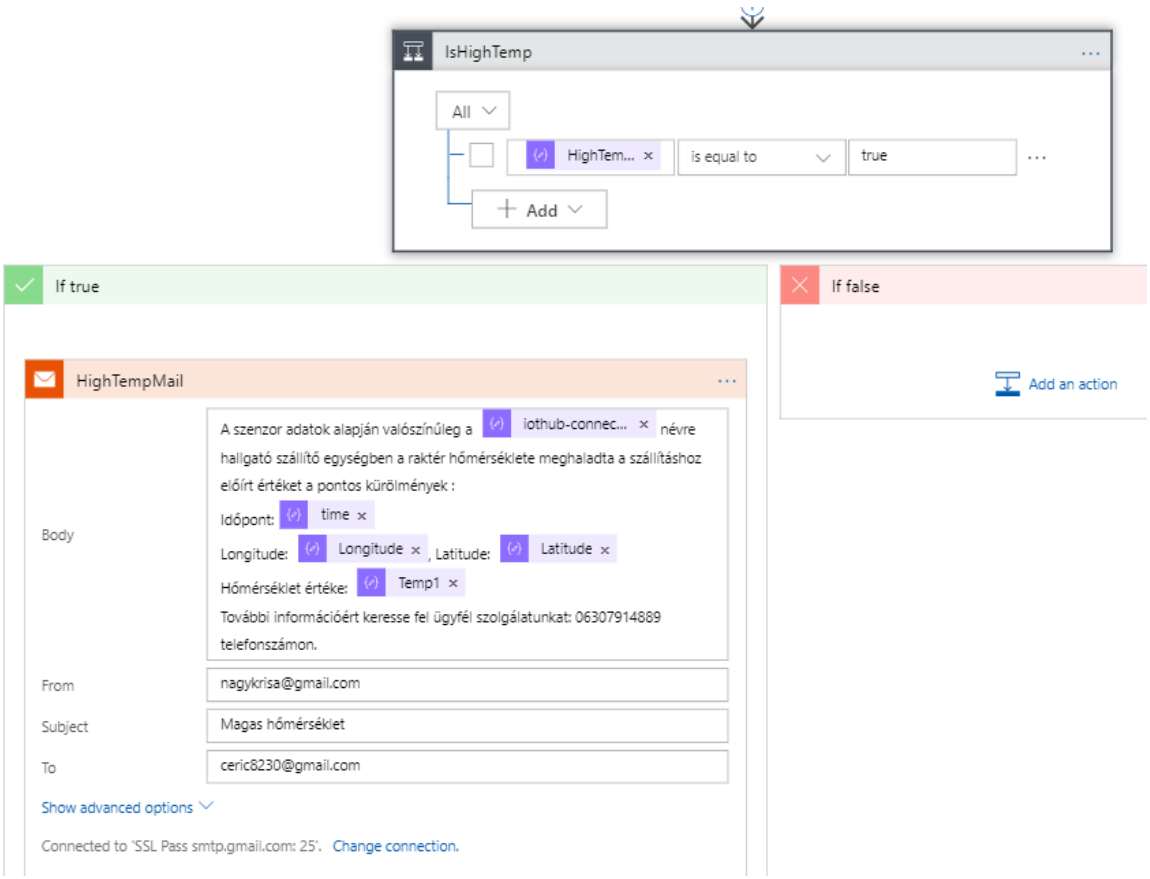
A forgatókönyveinket egy grafikus felület segítségével hozhatjuk létre, amely minimális programozói és az Azure szolgáltatások körültekintő ismerete mellett könnyen és gyorsan kezelhető. A Designerhez tartozik ezen felül egy forráskódos nézet, amelyen keresztül a grafikusan elkészített változat forráskódbéli megfelelőjét szerkeszthetjük. Az egyes forgatókönyveket egy beépített verzió kezelő segítségével is kezelhetjük, így a korábbi forgatókönyveink bármikor visszakereshetőek. A forgatókönyvekben használatos API összeköttetéseket egy külön felületen tarthatjuk karban. Az általam a grafikus tervezői felületen létrehozott forgatókönyvet az 5.6.2-es ábra ismerteti és a következők alapján működik.



5.6.2. ábra: Az általam elkészített forgatókönyv folyamat ábrája

A forgatókönyv első lépése mindig egy úgy nevezett Trigger esemény. A szolgáltatás vár, majd a Trigger esemény bekövetkeztekor az őt követő további lépések egymást követően hajtódnak végre. Az én Trigger eseményem egy Service Bus üzenetsorba érkező üzenet esetén aktiválódik. Tehát ha a telematikai eszközünk üzenetet küld az IoT Hub-nak, akkor az berakja ezt az üzenetet a Service Bus üzenetsorába, amely esemény hatására a Logic App elvégzi a forgatókönyvben leírtakat az üzeneten. Az üzenet Content névvel ellátott részét egy base64 dekódolóval olvassuk be majd egy JSON fájlba rendezzük. Ez a JSON objektum tartalmazza a telematikai eszköz által mért adatokat. A Content résszel megegyezően az üzenetünk Body névvel ellátott részét is egy JSON objektumba rendezzük. Ez az objektum az eszközünk

keltette riasztások feldolgozásához szükséges adatokat tartalmazza. Az üzeneteket feldolgozó lépéseket követően, egy feltételes útválasztás alapján a megfelelő eseményekről e-mailes értesítést küldünk a megadott címekre. Ha például magas a hőmérséklet, akkor a megfelelő feltétel igazra értékelése után, egy SMTP szolgáltatással való API kapcsolaton keresztül, az esemény paramétereivel ellátott e-mailt küldünk, amely folyamatot az 5.6.3-as ábra mutat be.



5.6.3. ábra: E-mailes értesítés küldése magas hőmérsékletről

Az általam megírt forgatókönyvben a magas hőmérséklet, baleset, illetve a rázkódás kiváltotta esetekről küldök e-amiles értesítést. Ha az eseményekkel kapcsolatos minden e-mailes tájékoztatást elküldtünk, a forgatókönyv következő lépése az üzenetek releváns részeinek mentése. Ehhez egy Azure Blob Storage API kapcsolatot használok, amelybe végezetül minden beérkező üzenet mentésre kerül.

Azure Storage

Az Azure Storage szolgáltatás használatához egy külön Storage felhasználói fiók létrehozása szükséges. A fiók létrehozásakor meg kell adnunk, hogy milyen típusú tárterületet szeretnénk az adataink eltárolásához használni. Ennek díjszabásbéli

következményei vannak, ennek megfelelően a legolcsóbb és legegyszerűbb Blob tároló típust választottam. Ennek a tárolónak az elérési műveletei viszonylag lassúak, ellenben kevesebbet kell értük fizetni és a tárolás havi díjazása is olcsóbb a többinél. A Blob tárolók karbantartását számos eszközzel támogatja az Azure irányítópultja. Hozzáférés menedzsmenttel kapcsolatos szabályokat és egyéb adat manipulációs és lekérdezésre alkalmas felületeket kínál. A tárolónk belső struktúrájának kiépítéséért mi vagyunk a felelősek. Ahogy a mappákat létrehozzuk, egy fa struktúrában tárolódnak el az egyes blobjaink, amelyek az adatainkat tartalmazó, levél csomópontokként jönnek létre.

Az Azure Storageben tárolt adatainkat bármikor feldolgozhatjuk további szolgáltatásaink segítségével, vagy bemeneti adatként szolgálhatnak más Azure szolgáltatásoknak.

Stream Analytics

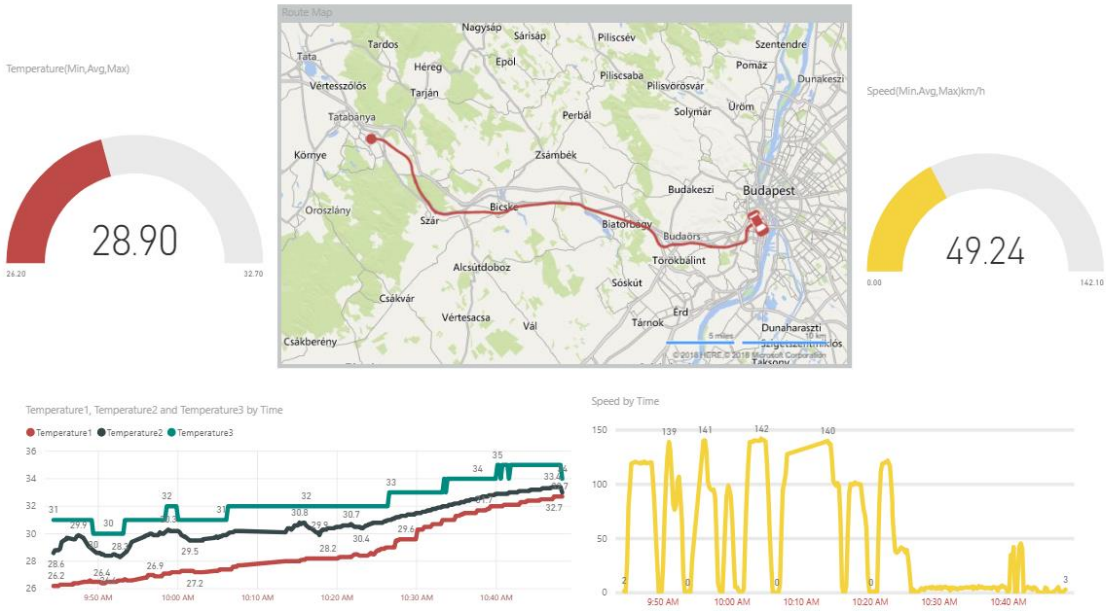
A Stream Analytics valós idejű adatfeldolgozást biztosít az adatfolyamaink számára. A korábban már említett erőforrás csoportunkba felvéve, az itt fellelhető szolgáltatásokkal működik együtt a Stream Analytics. A szolgáltatás irányítópultján úgynevezett munkafolyamatokat hozhatunk létre. Egy munkafolyamathoz három dologra van szükségünk: bemeneti forrásra, kimeneti forrásra, és az elvégzendő feladatra, lekérdezésre. Bemeneti forrásként felvehetjük az imént említett Blob tárolót, de ha valós idejű adatokat szeretnénk kezelni, kézenfekvőbb az IoT Hub-ba érkező friss üzeneteket kezelni. Kimenetként adatbázisok nagy választékából válogathatunk, ha az adataink rendezve, szűrve történő tárolása a célunk. Létezik viszont egy ennél számunkra hasznosabb kimenet. Egy PowerBI felhasználó fiókkal a Stream Analytics által feldolgozott adatokat egy úgynevezett PowerBI Datasetbe továbbíthatjuk kimenet gyanánt. Ezen Datasetek használatáról később szó lesz még. Az általam elkészített lekérdezés bemenetére az IoT Hub-ból érkező üzeneteket vettem fel és különösebb adatfeldolgozási műveletek nélkül a PowerBI Dataset végpontba irányítottam azokat.

Egy példán keresztül szeretném a szolgáltatás felhasználásának előnyeit ismertetni. Tegyük fel, hogy elindul a szállító egység és a telematikai eszközünk megkezdi az adatok küldését. A beérkező adatok közül a számunkra relevánsakat kiszűrve, továbbítjuk azokat a PowerBI Datasetbe. A Datasetbe érkező már feldolgozott adatokból valós idejű monitorozást valósíthatunk meg.

5.6.2 A felhő rétegen alapuló szolgáltatások

PowerBI Dashboard

Ahogy azt említettem, a Stream Analytics-en keresztül az adataink egy Power BI Datasetbe kerülnek. A Power BI egy üzleti analitikai szolgáltatás, amelynek segítségével adatainkat interaktív grafikonokon, diagrammokon jeleníthetjük és oszthatjuk meg. Az általam használt, egy ingyenes 60 napos próba verzió volt, amely az egyetemi e-mail cím segítségével volt elérhető. A Datasetben reprezentált adatainkat, tetszés szerint testre szabható grafikonok együtteseként, egy Dashboardon keresztül tehetjük megtekinthetővé. Az előre elkészített grafikonokba csak be kell húznunk a táblázatos formában elérhető adatainkat, amelyeken még egyéb adatszűréssel kapcsolatos műveleteket is végezhetünk. Például, ha szeretnénk csak a dél és kettő óra közötti adatokat felhasználni a grafikonokon akkor ennek megfelelően végezhetünk szűrést az adatokon. A Dashboardok összeállítása kézzel történik, de az adatok már valós időben maguktól is frissülnek a felületen az 5.6.4-es ábra szerint.

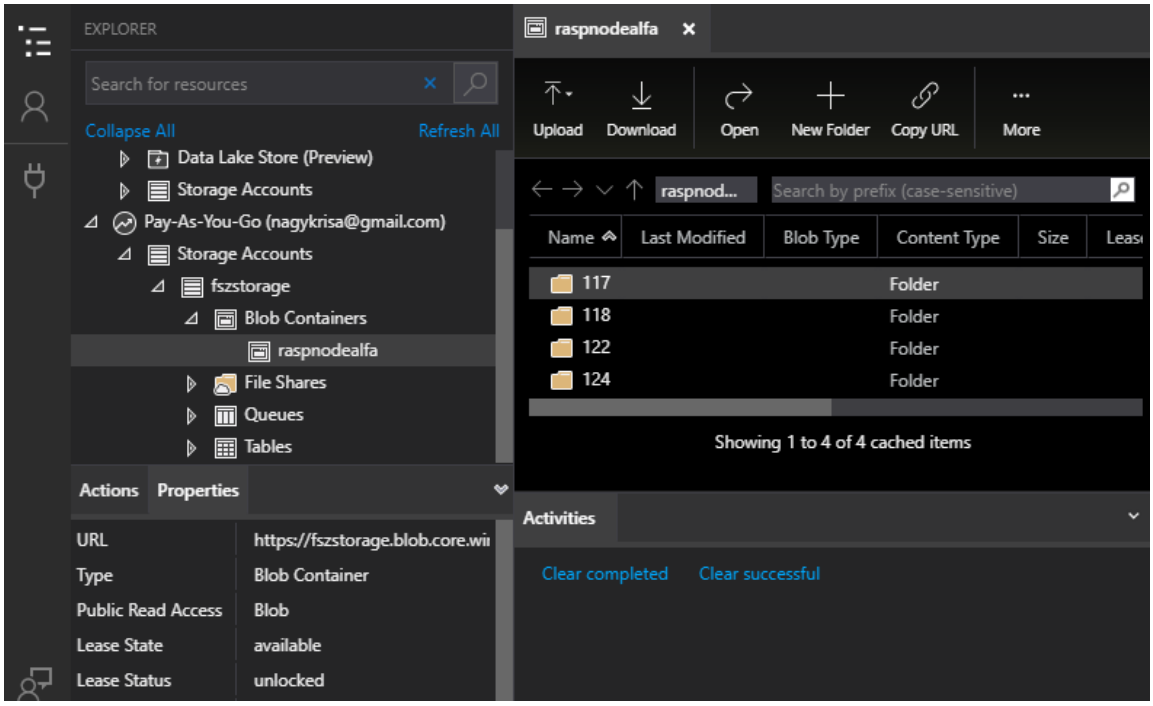


5.6.4. ábra: Általam elkészített Power BI Dashboard egy részlete

Blob tároló

A felhő szolgáltatások egy másik kimenetének végeredménye, a már korábban említett Blob tárolók láthatók az 5.6.5-ös ábrán. Ezeket a tárolókat több felhasználói felületről is elérhetjük. Ehhez egy magas szintű kezelési lehetőséget nyújt a Microsoft Azure Storage Explorer nevű asztali alkalmazás. Ennek segítségével karbantartási

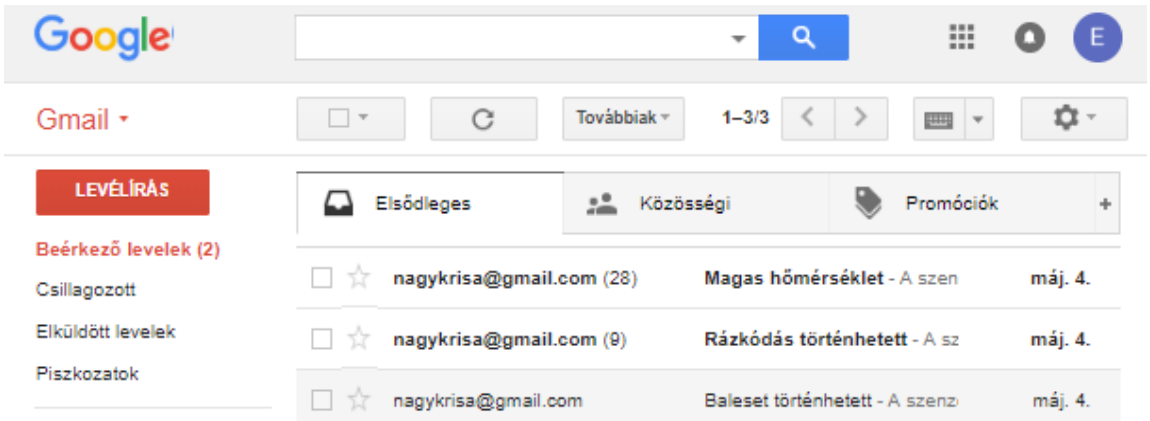
munkálatokat végezhetünk a tárolóinkon, lekérdezhetünk adatokat és egyéb adatbázis kezelőknél megszokott műveletet végezhetünk.



5.6.5. ábra: Naplózott adatok a Storage Explorerben naponkénti bontásban

E-mail jelzés

Ha már a szolgáltatások jelentette kimeneteknél tartunk, érdemesnek tartom bemutatni a prototípus rendszer működése során keletkezett e-mailes értesítéseket, amelyeket az 5.6.6-os ábrán láthatunk. A beérkezett üzenetek alapján láthatjuk, hogy mikor volt baleset, rázkódás vagy a megengedettnél magasabb a hőmérséklet, ezzel is bizonyítva a módszer működőképességét.



5.6.6. ábra: E-mailes értesítők a riasztásokról

5.7 A felhő réteg gyakorlati szoftveres megoldásai

A háttérrendszer tekintetében számos olyan megoldás ötlete felmerül, amely a prototípus elkészítése során vagy anyagi vagy erőforrásbéli okokból nem volt kivitelezhető, megoldható. A következő javaslatok mindegyike egy nagyobb számú szállító egységet feltételező eszközpark esetén lehet kifizetendő.

IoT HuB

Az IoT Hub esetén a korábban eszköz oldalon már ismertetett Cloud to Device típusú üzenetek felhasználási lehetőségeire itt már nem szeretnék bővebben kitérni. Ehelyett a megnövekedett telematikai eszköz szám jelentette feladatok kezeléséről szeretnék szót ejteni. Az első és talán legfontosabb kérdés az igénybe vett szolgáltatás költsége. Folyamatos, 12 másodpercenkénti üzenet küldéssel kalkulálva, eszközönként napi 7200 üzenet keletkezik. A kiszolgálható eszköz száma a díjszabási csomagonként a következők szerint változik:

- Standard1 csomag 55 eszköz ellátásához havi 21 euróért (0,38euro/eszköz)
- Standard2 csomag 833 eszköz ellátásához havi 210 euróért (0,25euro/eszköz)
- Standard3 csomag 41666 eszközhöz havi 2100 euróért (0,05euro/eszköz)

Ahogy láthatjuk, minél nagyobb számú eszközt szeretnénk kezelni, annál kedvezőbb egységárakat tapasztalhatunk. Ezért érdemes előre felmérni, hogy mekkora számú eszközparkra kell a rendszert kiépíteni. Ezenfelül ezekhez a szolgáltatás csomagokhoz már több végpont és útvonal is felvehető, amelyek az adatok választékosabb felhasználási formáit teszik lehetővé.

Service Bus

A Service Bus szolgáltatás díjazása alapvetően az áthaladó üzenetek számával arányosan skálázódik. Standard, illetve Prémium szintű csomagjai lehetővé teszik a Topicokban történő üzenet továbbítást, amely az egyes eszközöktől érkező üzenetek felcímkézésével újabb érdekes eset szétválasztási lehetőségeket rejtenek magukban.

Például a szállítmány típusának és az eszerint kinyert adatoknak megfelelően külön Topicokba kerülhetnek a fagyasztott árut kezelő eszközök adatai és külön topicba a gázolaj szállítmánytól érkezők.

Azure Storage

Az adatbázisok felhasználása területén még sok, gyakorlati megfontolásokat támogató lehetőség rejlik. Azure SQL és tábla alapú adatbázisok segítségével számos, a szállítmányozásban egyébként is már használatos adatbázist integrálhatunk a szolgáltatásainkba. Ezzel a módszerrel a már meglévő és működő rendszerek kiváló kiegészítő elemei lehetnek az általunk elkészítendő rendszernek.

Vegyük például a következő esetet. Van egy adatbázisunk szállítható termékekről, a rájuk vonatkozó szállítási feltételekkel és ezen feltételek értékhatáraival. Élelmiszer szállítmány esetén tegyük fel, hogy minden szállított élelmiszernek adott egy szállítási hőmérséklet, például paradicsom konzerv maximum 40 fok, minimum -5 fok. Amikor elindul a szállító egység az élelmiszer rakománnyal akkor az adatgyűjtő eszközünk a szállítmány áruit nyilvántartó SQL táblából értesül ezekről a szállítási értékhatárokról és csak ezeknek megfelelően küld riasztásokat.

Tehát korábbi SQL táblákat felhasználva, egy eszköz innentől kezdve már csak a szállított rakománynak megfelelően fog riasztásokat küldeni. Ilyen és ehhez hasonló megfontolásokkal élve számos lehetőséget rejtenek az adatbázisaink.

Logic App

A már megismert szolgáltatás segítségével további tetszőleges bonyolultságú forgatókönyveket hozhatunk létre események bekövetkeztét nyomon követve.

Például szállítmányok indulásának és érkezésének kapcsán SMS-ben értesíthetjük a rakodást végző munkásokat, hogy időben álljanak készen a feladatok elvégzéséhez. További példaként, egy biztonsági kamera segítségével felvételt készíthetünk a nyitott raktér során történő tevékenységekről. Ezt az adathalmazt tipikusan pazarló és túl költséges lenne azonnal közvetíteni egy felhő szolgáltatásnak, de ha csak a felvétel készítésének tényével vagyunk tisztában, a szállító egység logisztikai központba való érkezésekor, megbízható WiFi kapcsolaton keresztül a videó már feltölthető egy erre alkalmas adatbázisba.

Ilyen és ehhez hasonló egyéb komplexebb forgatókönyvek leírásához használható fel a Logic App, természetesen figyelembe véve annak anyagi költségeit, amely már néhány magasabb üzleti és logisztikai szolgáltatás alkalmazása esetén is komoly anyagi fogyasztást eredményezhet.

Web Apps Service

Egy saját honlap készítése ma már nem számít nagy újdonságnak, mégis az Azure szolgáltatásait integrálva, hatékony felhasználói portált készíthetünk. A regisztrált felhasználók a honlapon keresztül vezethetik fel szállítmányozási igényeiket. A beérkezett igények alapján történne az eszközpark logisztikai elosztása, illetve a szállítmányozás folyamatának tervezése. Ugyancsak a regisztrált felhasználóknak az általuk kért szállítmány útja során keletkező adatokat ebbe a honlapba ágyazott grafikonokon jelenítenénk meg. A honlapon keresztül a korábbi szállítmányok adatai is elérhetővé tehetők az adatbázisokból a felhasználók számára, későbbi feldolgozásra, vagy csak egyszerű naplózás céljából.

Egy honlap készítése számos további felhasználási lehetőségeket nyújthat még, akár a felhasználók vagy a szolgáltatás igénybevételi szokások elemzését.

Functions

Az Azure Functions segítségével egyszerűen futtathatunk kisebb kód részleteket, függvényeket a felhőben és lehetővé teszi a kiszolgáló nélküli alkalmazások fejlesztését. Ezeknek a kódoknak a futtatásához nem kell külön infrastruktúrális vagy alkalmazásbeli problémákkal foglalkoznunk.

A futtatások díjazása óránként történik, tehát csak az aktuálisan futott függvény órákért kell fizetnünk. A Logic Apphoz hasonlóan itt is képesek vagyunk az egyéb felhőszolgáltatásaink közötti interakciók, funkciók leírására és együttműködésük megteremtésére. A Functions szolgáltatásainak tehát egy jelentős előnye a Logic App-al szemben a kedvezőbb díjazási feltételek, ezért szintén előre mutatónak találom alkalmazását a szállítmányozási rendszer felhőszolgáltatásaiban.

5.8 Költség és fogyasztás becslése

Egy rövid számolás erejéig szeretném összevetni, hogy mennyibe is kerülne a szolgáltatás 50, 800, 40 000 darab szállító egységből álló flotta szervezéséhez. Ehhez először az eszköz oldali hardver költségeket kell felmérni egy szállító egységre.

Ha eszközök hálózatában gondolkodunk az általam felhasznált szenzorok összértéke 15 000 Ft volt. Nyilván szállítmány függő a szenzorok használata, de a beépített szenzorok árára egy jó felső közelítés a 20 000 Ft. A szenzorokból adatot olvasó eszközök hálózatával számolva 4 NodeMCU ára körübelül 10 000 Ft. A

tápellátás tekintetében ezekhez az eszközökhöz egy meghajtásról táplált komolyabb külső akkumulátor elegendő, amely árát 15 000 Ft-ra becsülöm. Ennek fényében 1 szállító egységhez tartozó eszköz csomag körülbelüli maximális ára 45 - 50 000 Ft-nál megáll.

A továbbiakban a felhőszolgáltatás költségeit fogom felbecsülni 1 hónapra 1 eszközre mérve. A Stream Analytics szolgáltatás folyamatos működés mellett eszközönként 20 000 Ft, amely bőven felső becslés mivel biztos, hogy egy hónapon keresztül nem kell folyamatosan, valós időben adatot közvetítenünk. A Logic App szolgáltatás használata túlzóan költséges lenne a megoldásban történő felhasználáshoz, amely funkcióit egyszerűbben is meg tudunk oldani az Azure Functions segítségével. A Service Bus költségei havi szinten számolandó akárcsak az IoT Hub-é vagy az Azure Storage-é.

Így egy durva közelítő becsléssel az alábbi árakhoz jutottam:

- 50 eszköz telepítése 2,5 millió + havi 0,5 millió Ft üzemeltetés
- 800 eszköz telepítése 40 millió + havi 4,5 millió Ft üzemeltetés
- 40 000 eszköz telepítése 2 000 millió + havi 120 millió Ft üzemeltetés

A növekvő eszköz számmal arányosan egyre inkább kifizetődőbb a saját eszközös megoldások gyártása, a készen kapható termékekkel szemben, így tovább csökkenthető az eszközök telepítésének ára. Mindemellett az Azure szolgáltatások költségei is jelentős mértékben csökkenthetők, ha nagy mértékben vesszük őket igénybe. Az általam végzett számítások csak nagyságrendben közelítik meg a pontos értékeket, ezek pontos ára nagyban függ az igénybe vett szolgáltatásoktól és azok használatának mértékétől, valamint az eszközök beépítési, illetve üzemeltetési költségétől is.

6 A prototípus rendszer tesztje egy példán keresztül

Az eddig alkotórészenként ismertetett rendszerem működéséről szeretnék most egy összefoglaló képet adni a könnyebb megérthetőség, átláthatóság érdekében, mégpedig úgy, hogy működése közben egy példán keresztül vezetem végig a történéseket és az adatok útját.

Ennek az ismertetéséhez, a követhetőség érdekében a már korábban ismertetett architektúra alapján három elkülöníthető rétegre bontom a bemutatást:

- Eszköz réteg - telematikai eszköz áttekintése
- Felhő réteg – Azure szolgáltatások áttekintése
- Szolgáltatás réteg – a felhasználóknak nyújtott szolgáltatások áttekintése

A teljesen inaktív eszköz működésbe hozásához, indulásra kész állapotba helyezéséhez szükséges úgynevezett nulladik lépések elvégzése az eszköz rétegben.

6.1 Eszköz réteg

Indulás előtt a mobil telefonomon célszerű bekapcsolni a megosztható WiFi hozzáférési pontot, amelyre csatlakozva a telematikai eszköz eléri a felhőszolgáltatásokat majd. Ha ezzel megvagyunk, a telematikai eszközünket a külső akkumulátorra dugva meg is kezdődhet az adatok gyűjtése. Ezt a belső folyamatot szeretném részletesen bemutatni. Az áramforrásra csatolást követően az eszközünk operációs rendszere bebootol és rögtön ezután elindul a szolgáltatásunk, amely biztosítására a pm2 program szolgál. A pm2 szolgáltatás az általunk magadott szabályok szerint menedzseli, futtatja a neki címzett Node.js programunkat. Természetesen az eszköz ezzel párhuzamosan csatlakozik az elérhető telefonos WiFi hotspot-ra is.

A Node.js kódunk futása során, az eszköz internetes hozzáférést feltételezve megpróbál csatlakozni az IoT Hub-hoz, majd megkezdődik az üzenetek küldése. Amennyiben nincs elérhető hálózat, az eszköz lokális üzenetsorokban várakoztatja a küldésre szánt csomagokat és amint lehet, elküldi őket. Ekkor már közben elindulhatunk azon az útvonalon, amely egy szállítmányozási folyamatot szimulál. Miközben sétálunk

vagy pozíciót változtatunk valamilyen jármű segítségével, az eszköz folyamatosan gyűjti és küldi a szenzorokon keresztül mérhető adatokat.

Észleli a rázkódásokat, fordulásokat, gyorsulásokat, nyomon követi a pillanatnyi hőmérséklet, páratartalom és légnyomás értékeket, valamint, abban az esetben, ha a műholdas adatok biztonsággal elérhetőek, tudjuk a pontos földrajzi szélességi és hosszúsági koordinátáinkat, a tengerszint feletti magasságunkat, pillanatnyi sebességünket és a pontos időt. Sajnos a GPS modul jellegéből adódóan a földfelszín alatti közlekedés tényét, illetve a vastagabb falú épületekben történő működés során a GPS adat érzékelését nem tudjuk garantálni. Így például egy hosszabb alagúton történő áthaladás esetén, ha nem megoldott a jel átvitele az alagútban, akkor pár GPS koordináta elveszhet. Ez a jelenség viszont áthidalható egy számolással, ha pontos adataink vannak az alagútba való be és kilépés koordinátájáról, illetve a pillanatnyi sebességről, amelyet a szállító jármű CAN-bus-áról elérhetünk.

A mért adatok a belőlük származtatott riasztásokkal együtt kerülnek üzenet csomagokba. Ezek a csomagok helyileg mentésre kerülnek, illetve megkezdik útjukat a felhasználók felé.

6.2 Felhő réteg

Az adatok felhasználókhoz való eljutását viszont megelőzi a felhőszolgáltatásokon keresztül történő feldolgozásuk. Az IoT Hub-ba érkező adatok a magadott útvonalakon keresztül eljutnak az általunk definiált végpontokhoz. Esetemben az adatok, beérkezésüket követően a Service Bus üzenetsorába kerülnek. Az üzenetsorból a Logic App szolgáltatás veszi ki a beérkezett üzeneteket. Minden egyes üzenet esetén megvizsgáljuk, hogy van-e riasztani való esemény. Ha nincs riasztás, egyszerűen naplózzuk, mentjük az üzenetsorból kivett üzenetek tartalmát egy Blob tárolóban. Ha riasztást észleltünk, akkor a megadott címen értesítést küldünk az eset bekövetkeztéről.

Például séta közben megbotlunk és a hátizsákunkban lévő eszközt kisebb rázkódás éri. Ekkor a megadott e-mail címen 12 másodperces késéssel megjelenik a rázkódás ténye és az, hogy pontosan mikor történt az eset, melyek voltak az egyéb fizikai jellemzők, mint pozíció vagy a gyorsulás mértéke. A rázkódások besorolásához egy külön kisebb kutatás tartozott a munkám során. Ennek kapcsán először a közúti közlekedés során mérhető gyorsulás értékeknek jártam utána. Azt tapasztaltam, hogy a

0 és 0,2 G közé eső gyorsulás értékeket célszerű figyelmen kívül hagyni, hiszen az e két érték közé esők egy sima vontató jármű közlekedése során is felmerülnek. Ehhez egy rövid számítás alapján jutottam, amelyben egy interneten olvasott adattal éltem, miszerint egy rakomány nélküli vontató jármű 16 másodperc alatt gyorsult fel 100km/h-ra. Az eszközön rázkódásokat szimulálva úgy becsültem, hogy a 0,2 G fölötti gyorsulás érték, már megfelel egy erős fékezésnek vagy út hibába futásnak, amely káros lehet nem megfelelően rögzített szállítmány esetén. Az 1G feletti értékekhez a baleset tényét rendeltem. Ekkora gyorsulás felett már nem beszélhetünk természetes fékezésről.

Tehát a szolgáltatásom a rázkódás, baleset és magas hőmérséklet riasztásokat kezelte a működése során. Az adatok egy másik utat bejárva a PowerBI Datasetekbe kerülnek, ahonnan valós idejű monitorozáshoz használjuk fel őket. Ehhez, ahogy már említettem a Stream Analytics adatfeldolgozó szolgáltatáson haladnak át az üzeneteink, amely a bemenetére kapott üzeneteken elvégzi a definiált adatműveleteket, majd a felhasználható adatokat továbbítja a kimenetére a szolgáltatás rétegbe.

6.3 Szolgáltatás réteg

A szolgáltatás réteg alatt azon felhasználói felületek összességét értem, amelyeken keresztül az adatainkból előállított felhasználható információt megjelenítjük és elérhetővé tesszük az abban érdekeltek számára, akár csak monitorozás, esemény kezelés vagy későbbi feldolgozás céljából.

Ezek a szolgáltatások egy új szintre emelik a szállítmányozó cégek és a szolgáltatásaikat igénybe vevő felhasználóik közötti bizalmat, hiszen a felhasználók a számukra lényeges folyamatok legapróbb részleteibe is betekintést nyerhetnek. De nézzük meg pontosan, hogyan is történik ez. A felhő rétegig elérve az eszközünkből kinyert adatokból a számunkra releváns és felhasználandó információt szűrtük ki. A további feladatok az információ megjelenítése köré rendeződnek. Az e-mailes értesítések kezelését röviden már bemutattam.

Most egy riasztás esetének részletes útvonalát írom le. Tegyük fel, hogy a hőmérséklet az érzékelő körül 28 fokra emelkedik. A kódunk kiolvassa az adatot és igazzá értékeli a feltételt miszerint a hőmérséklet értéke meghaladta a 27 fokot. Ez az információ egy boolean érték formájában kerül csatolásra az elküldendő üzenethez. Az IoT Hubba érkező üzenet először a Service Bus üzenet sorába kerül, ahonnan a Logic App szolgáltatás emeli ki. Az üzenetet elemeire bontása után detektáljuk az igazra

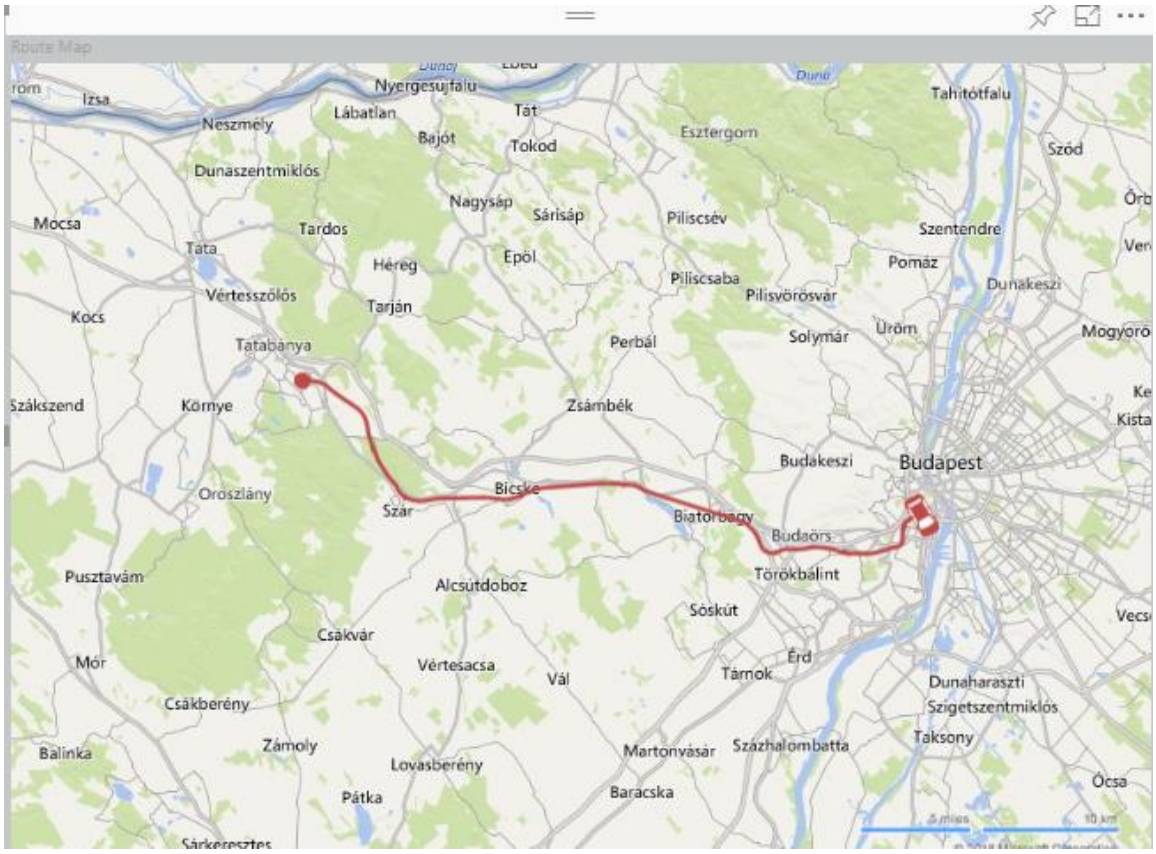
értékelt magas hőmérséklet riasztást és ennek következtében megkezdődik az értesítés. Saját Gmail fiókomhoz kapcsolódva, azon keresztül a megadott e-mail címekre elküldünk egy e-mailt a pontos hőmérséklet értékkel és a riasztás szövegével. Így történik az adatok alapján a felhasználók riasztásokról történő értesítése és tájékoztatása e-maileken keresztül.

A továbbiakban a szállítás során monitorozott adatok útját fogom bemutatni az eszköztől egészen a szolgáltatásig. Az eszköz réteg szenzoraiból kiolvasott adatokat egy üzenetbe csomagoljuk. Az eszköz rétegből küldött üzenet a tartalmával együtt a felhő rétegbe érkezik, azon belül is az IoT Hub-hoz. Innen a Stream Analytics szolgáltatásunk veszi ki az üzenetekben fellelhető, mért adatokat tartalmazó részt. Az adatainkat tartalmazó részt az esetünkben mindenféle szűrést nélkülözve küldjük tovább egy PowerBI Datasetbe. A Datasetbe érkező adatokat táblázatos formában érhetjük el. A táblázatunk oszlopai az általunk elnevezett adatokat hordozó változók nevét veszik fel. A táblázatunk soraiba az egyes üzenetek tartalma kerül idő folytonos sorrendben. Tehát minden újonnan beérkező üzenetből származó adat bekerül a táblázatba az őt megelőző üzenet adatai után. A Dataset táblázatából először grafikonok formájában vizuálisan megjelenítjük az adatainkat. Ezt például megtehetjük úgy is, hogy az általunk kiválasztott grafikon tengelyeihez rendeljük a táblázatunk ábrázolni kívánt oszlopait. Egy több grafikonból összeállított felületet úgynevezett „jelentés” -ként menthetünk el. Ezt a jelentést közzé tehetjük a felhasználóink számára, ekkor már egy Dashboardról beszélünk, amelyen szereplő grafikonok adatai valós időben frissülnek, amíg az adatgyűjtés folyamata zajlik.

6.4 A gyűjtött adatok bemutatása és részletezése

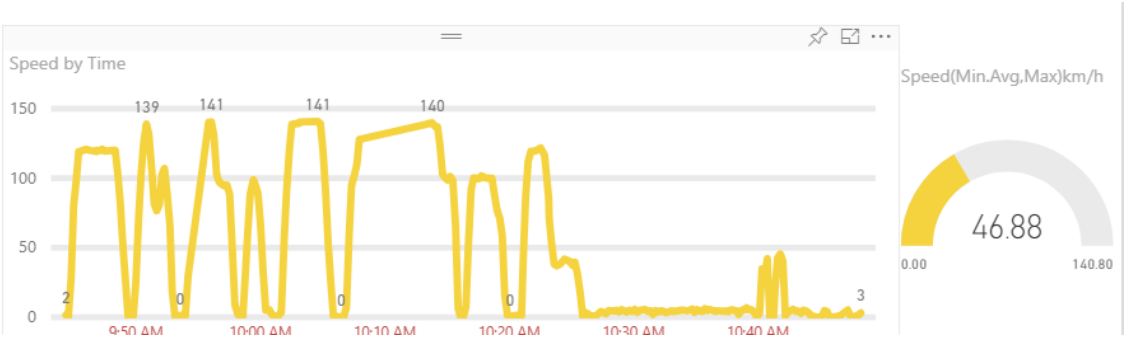
Jelen fejezet célja bemutatni az általam gyűjtött adatok halmazából készített grafikus megjelenítési felületeket. Ezek lényegében a PowerBI Dashboardjában megjelenített diagrammok és grafikonok összessége.

A GPS modulból gyűjthető adatokkal kezdve, azok közül is talán a legfontosabbat kiemelve, a földrajzi szélességi és hosszúsági koordináták ábrázolását szeretném bemutatni a 6.4.1-es ábrán.



6.4.1. ábra: Útvonal a földrajzi koordinátákból

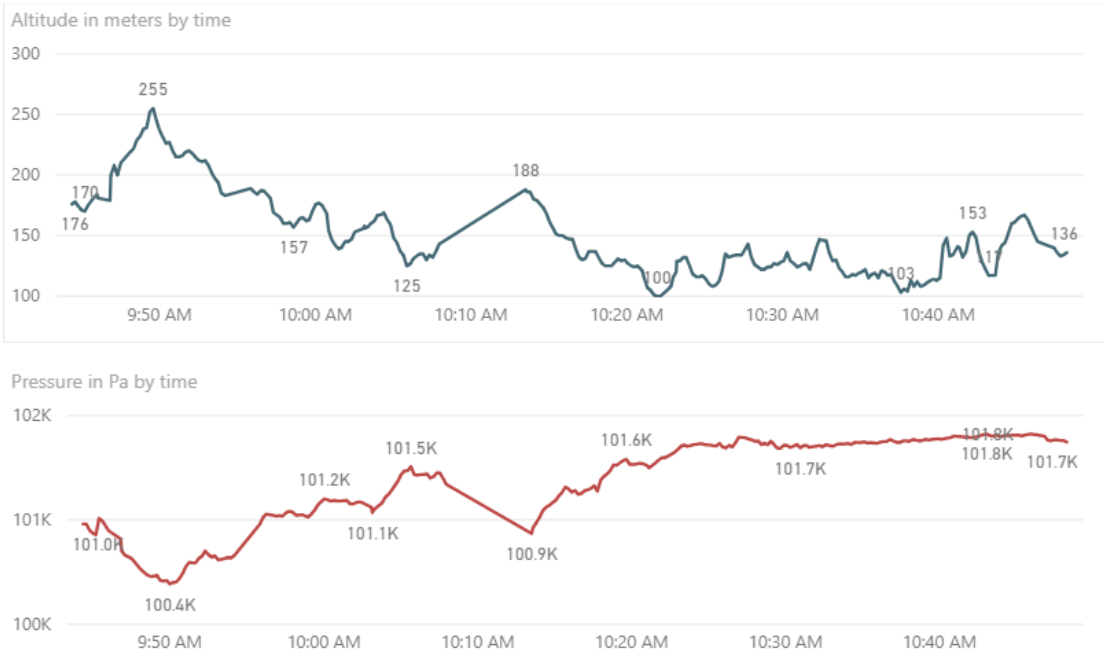
A diagramunk a megadott időrendi sorrendben összeköti a szélességi és hosszúsági koordináták által kijelölt útvonalat. A piros pont jelzi az út kezdetét és az felülnézeti autóikon jelzi az aktuális pozíciót. Az útvonalon elején személyvonattal közlekedtem a Tatabánya, Kelenföld útvonalon, amelyet egy séta követett a 19-es villamos megállójáig, innen villamossal utaztam a Kosztolányi Dezső térig, ahonnan ismét gyalog haladtam tovább a célpontomig. A következő érdekes GPS adatunk a sebesség mért értékei az idő függvényében.



6.4.2. ábra: Az út során keletkezett sebesség adatok

A 6.4.2-es ábra bal oldalon az idő sebesség függvényt látjuk, amelyen jól kivehető a vonattal való közlekedés. Az egyes állomások között 100, 140 km/h-val közlekedtünk és jól kivehetők az állomások is ahol 0-ra csökkent a sebesség. A vonatról leszállva kivehető a séta 4-5 km/h-ás sebessége. Ezt követően ismét felfedezhető a villamosra jellemző 40-50 km/h sebesség és a megállók jellegzetes formája. A villamosról leszállva további séta következett, amelyet az adatok is alátámasztanak. A jobb oldali diagram két szélső értéke a minimális 0, illetve a maximális 140km/h-át jelzi, a köztes érték pedig az átlagsebesség, amely 46,88km/h körül alakult.

A GPS modulból továbbá tengerszint feletti magasság értékeink is vannak, amely mért értékeit méterben ábrázoljuk az idő függvényében a 6.4.3-as ábra szerint. Ennek bemutatását érdekesebbnek tartom összekötni a BMP180-as szenzor által mért légnyomás értékekkel.

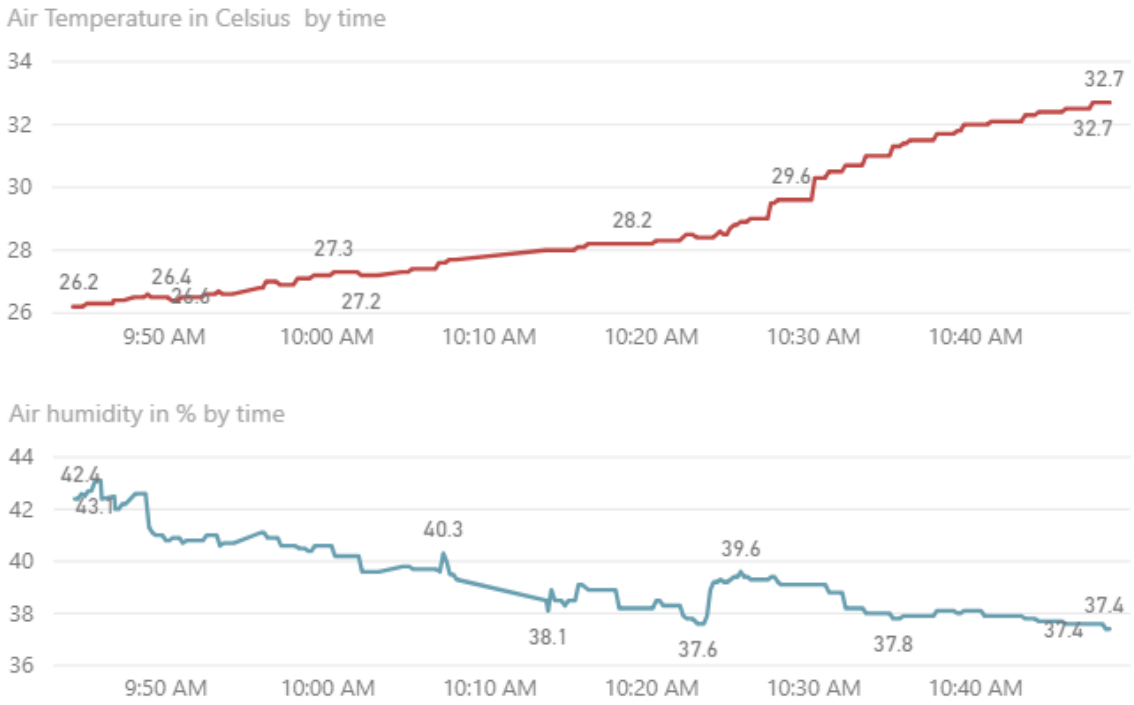


6.4.3. ábra: Az út során mért tengerszint feletti magasság és légnyomás értékek

A tengerszint feletti magasság értékeken jól megfigyelhető a Tatabánya környékén mért 167 méter körüli értékek. Az úton előre a Dunántúli-középhegységen áthaladva egy kisebb emelkedést láthatunk 255 méteres maximális magasságig, amely végül kisebb lankákon át Budapestet elérve közelíti a 136 méteres magasságot.

Amiért érdemesnek találtam a légnyomás adatokat itt feltüntetni az erősen szembetűnő. Megfigyelhető, hogy a magasabb földrajzi helyeken a légnyomás értéke csökken, ahogy ez elvárható, míg az alacsonyabb területeken a ránk nehezedő légoszlop nyomása megnövekszik. A mért légnyomás értékek jól láthatóan 100 kPa és 102kPa között alakultak, amely a meteorológiailag mért érték tartományba esik.

A következő szenzor adatok, amelyeket a 6.4.4-as ábrán látunk a DHT22-es páratartalom és hőmérséklet szenzorból származnak.

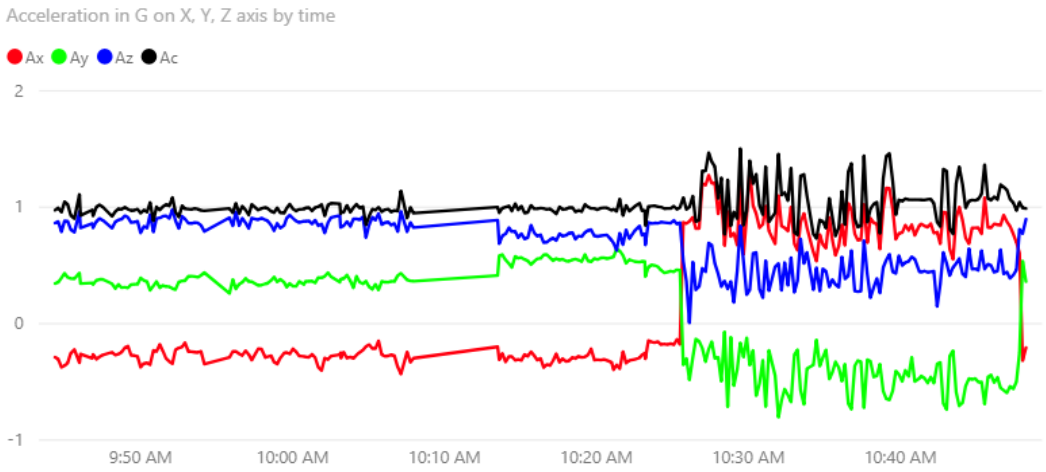


6.4.4. ábra: Az út során mért hőmérséklet és páratartalom értékek

Itt is megfigyelhető, hogy a mért adatok korrelálnak egymással. Mégpedig az alacsonyabb hőmérsékleten a levegő relatív páratartalma magasabb, és ahogy egyre inkább felmelegíti az eszközünk működése során a hordozódobozát, úgy figyelhető meg a levegő relatív páratartalmának csökkenése is.

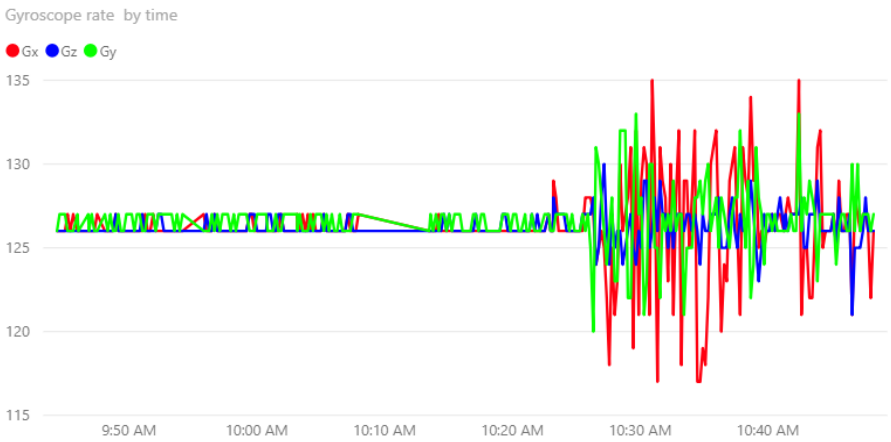
A továbbiakban a GY-87-es szenzor modulhoz tartozó adatokról lesz szó azon belül is az MPU6050-eshez tartozókról, mivel a BMP180-as értékeit már bemutattam.

Az MPU6050-es szenzor alapvetően gyorsulás értékeket mér az X, Y és Z tengelyen, valamint giroszkóp adatokat. A gyorsulás adatok a 6.4.5-ös ábrán láthatjuk, amelyek a következőképp alakultak.



6.4.5. ábra: Az út során az X, Y, Z tengelyen mért gyorsulás adatok

Ax, Ay, Az görbék az X, Y, Z tengely menti gyorsulást mutatják piros, zöld, kék vonallal. A fekete görbe az eredő gyorsulás, amely a három érték négyzet összegének a gyöke. Ennek megfelelően a földi nehézségi gyorsulás értékéhez hűen mindig 1 G körüli értéknél található a fekete görbe. Ami a tengelyek alapján látható, hogy az eszköz a vonat útja során viszonylag egyenletes, rázkódás mentes környezetnek volt kitéve. Amikor az eszközt átraktam a táskámba látszik, hogy a tengelyek iránya felcserélődött, más orientáltságot vett fel, valószínűleg az élére lett állítva ekkor a doboz. Egy pontosan rögzített szenzor esetében az eszköz pontos orientáltságát is meg tudnánk mondani ekkor. A séta közben már sokkal nagyobb rázkódás, gyorsulás értékek láthatóak a lépések következtében. A giroszkóp hasonló eseményekről tanúskodik a 6.4.6-os ábrán.



6.4.6. ábra: Az út során a giroszkópból mért adatok

7 Konklúzió

Szakdolgozatom bevezetésében azzal a feltevéssel éltem, hogy szükséges lenne napjaink közúti szállítmányozásának működését egy olyan IoT rendszerrel kiegészíteni, amely támogatja a szállítmányok és szállító egységek nyomon követhetőségét és folyamatos monitorozást tesz lehetővé a szállítás útja során. A rendszer létrehozásának szükségét a meglévő rendszerek hiányosságai, illetve az iparban megjelent új technológiák igényei alapozták meg.

A rendszerhez a már létező IoT és kiberfizikai rendszerek architektúráját vettem alapul. Ezek alapján építettem fel saját okos szállítmányozási rendszerem architektúráját, amely komponenseit magam választottam meg. Minden komponens megválasztása esetében odafigyeltem a globális rendszer skálázhatóságának feltételére és alkalmazhatóságára. Az általam összeállított telematikai eszköz és a rajta futó adatgyűjtő kód alkalmas a mozgó szállító egységekből történő valós idejű adatgyűjtésre. Az általam felhasznált és személyre szabott felhőszolgáltatások együttműködése eredményesnek bizonyult a kigyűjtött adatok feldolgozásában és az azokból történő tettre fogható információ előállításában. A feldolgozott adatokat pedig további szolgáltatások bevonásával alakítottam a felhasználók számára is alkalmas megjeleníthető információvá.

Így az IoT rendszerem nyújtotta szolgálatások segítségével felügyelhetjük, és valós időben monitorozhatjuk a szállítmányainkat és ezzel együtt a szállítmányt érő eseményeket. Rendszerem elkészítése során olyan valós felhasználói igényeket is figyelembe vettem, amelyeket a szállítmányozás folyamatában jártas vagy részt vevő ismerőseim tapasztalatai alapján keletkeztek, vagy az interneten fellelhető cikkek alapján megoldandónak találtam.

Ennek fényében az általam tett valós alkalmazási kiegészítésekkel élve működésre alkalmasnak találtam az elkészített rendszerem. Az IoT rendszerem támogatja a szállítmányozás folyamatának felügyeletét.

A jelenlegi rendszeremet a jövőben számos olyan egyéb funkcióval és szolgáltatással lehetne még kiegészíteni, amely tovább növelné annak értékét. Az érték tovább növelése alatt olyan ötletekre gondolok, amelyekkel kiegészítve tovább

növelhető a rendszer hatékonysága, felhasználhatósága és az általa felügyelt folyamatok optimalizálása.

Kezdetben egy webes felhasználói felület nagyban megkönnyítené a felhasználók számára történő információ áramoltatást. A weblapba beépített nézeteken keresztül tekinthetnék meg a felhasználók a hozzájuk tartozó szállítmányokat jellemző adat sorokat. A honlap szintén alkalmas lehet a szállítmányozásból adódó rendelési, üzletkötési és logisztikai feladatok lebonyolítására. Egy külön weblap készülne a szállítmányozást igénybe vevő ügyfeleknek és külön felületen lehetne magát a rendszert érintő feladatokat is ellátni.

Egy másik tovább fejlesztési irány lehet az adatok feldolgozása. Nagy mennyiségű értékes adatunk van, amely feldolgozásával kinyerhető többlet információ még inkább alkalmas a hatékonyabb munkafolyamat végzéshez és ütemezéshez, valamint a további optimalizáció elősegítéséhez. Például mélytanuló algoritmusokat alkalmazva következtetéseket vonhatunk le az egyes szállító egységekhez tartozó rázkódás adatokból. Ezek alapján előre jelezhetjük, ha a szállító egység felfüggesztése elhasználódik, vagy kiszűrhető az agresszív vezetési stílus, nem megfelelő szállítmányozási folyamat.

Globális megoldásokban gondolkodva, rendszerek rendszereként képzelhető el okos szállítmányozási rendszerek összehangolt működése. Ezek a rendszerek egymással megosztott információkat felhasználva optimalizálják saját működésüket így elérve egy globális optimumot, hatékonyságot. Például a globálisan optimális útbejárás és üzemanyag fogyasztás érdekében vállalatok egymás között adhatnak át szállítmányokat meghatározott áru átrakodásra alkalmas pontokon, így minimalizálva az üres raktérrel közlekedő szállító egységek útvonalát.

Ezen megfontolásoknak további környezeti terhelést csökkentő vonzata is van, amely támogatja a szállítmányozás folyamatának környezetkímélőbb megoldását.

Irodalomjegyzék

[1] Magyar Szállítmányozók Szövetsége: Magyar Általános Szállítmányozási Feltételek http://www.szallitmanyozok.hu/dokumentumok.php (2017. május 30.)

[2] Központi Statisztikai Hivatal: Szállítási teljesítmények, közúti közlekedési balesetek 2017. IV. negyedév, https://www.ksh.hu/docs/hun/xftp/gyor/sza/sza1712.pdf (2018. március 2.)

[3] Bergenhem, Carl, Erik Hedin, and Daniel Skarin. "Vehicle-to-vehicle communication for a platooning system." *Procedia-Social and Behavioral Sciences 48 (2012): 1222-1233.*

[4] Logisztika.com: „Ezért van ennyi kamion az utakon", http://logisztika.com/ezert-van-ennyi-kamion-az-utakon/ (2017. november 20.*)*

[5] Thompson, Scott E., and Srivatsan Parthasarathy. "Moore's law: the future of Si microelectronics." *Materials today 9.6 (2006): 20-25.*

[6] Lee, Edward A. "Cyber physical systems: Design challenges." *Object oriented real-time distributed computing (isorc), 2008 11th ieee international symposium on. IEEE, 2008.*

[7] Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems, 29(7), 1645-1660.*

[8] Husi Géza, Ipar 4.0, https://www.researchgate.net/profile/Geza_Husi/publication/301607839_Industry_40_Hungarian/links/586762ea08aebf17d39c7d98/Industry-40-Hungarian.pdf (2016. november 16.)

[9] Lee, J., Bagheri, B., & Kao, H. A. (2015). A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters, 3, 18-23.*

[10] Maksimović, M., Vujović, V., Davidović, N., Milošević, V., & Perišić, B. (2014). Raspberry Pi as Internet of things hardware: performances and constraints. design issues, 3, 8.

[11] Microsoft Azure: IoT Hub Documentation, https://docs.microsoft.com/en-us/azure/iot-hub/ (2018. április 27.)

[12] Microsoft Azure: Storage Documentation, https://docs.microsoft.com/en-us/azure/storage/ (2018. április 5.)

[13] Microsoft Azure: Stream Analytics Documentation, https://docs.microsoft.com/en-us/azure/stream-analytics/ (2018. március 27.)

[14] Microsoft Azure: PowerBI Documentation, https://docs.microsoft.com/en-us/power-bi/power-bi-overview (2018. május 7.)

[15] Microsoft Azure: Service Bus Documentation, https://docs.microsoft.com/en-us/azure/service-bus-messaging/ (2017. december 12.)

[16] Microsoft Azure: Logic App Documentation, https://docs.microsoft.com/en-us/azure/logic-apps/ (2018. január 12.)