



MATEMATIKAI ÉS INFORMATIKAI INTÉZET

Menedzselhető beléptető rendszer fizikai megvalósítása

Készítette

Jámbor Alexandra

Programtervező Informatikus Bsc.

Témavezető

Dr. Tajti Tibor

Egyetemi Docens

EGER, 2023

Tartalomjegyzék

Bevezetés	4
1. A rendszer bemutatása	5
1.1. Internet of Things (IoT)	5
1.2. Webservice	6
1.2.1. Beüzemelés	6
1.3. Azonosítás	7
2. Teljesen elosztott megoldás	8
2.1. Technológiák	8
2.1.1. RFID	8
2.1.2. L293 D	9
2.1.3. ESP8266	10
2.2. RFID elérés	12
2.3. Rest elérés	13
2.4. Zárak vezérlése	15
2.5. Tesztelés	16
2.6. Lehetséges problémák	17
3. Csillagpontos megoldás	18
3.1. Technológiák	18
3.1.1. SSH	18
3.1.2. Raspberry PI	19
3.1.3. Node-RED	20
3.1.4. Arduino	21
3.1.5. MQTT	24
3.2. Rest elérés	24
3.3. Tesztelés	25
3.3.1. Mocha	25
3.3.2. should.js	26
3.3.3. Flow tesztelés	26
3.4. Lehetséges problémák	27

4. A két megoldás összehasonlítása	28
5. Továbbfejlesztési lehetőségek	30
5.1. Egyéb megvalósítási lehetőségek	31
Összegzés	35
Irodalomjegyzék	37

Bevezetés

Minden azzal kezdődött, hogy jelentkeztem az Eszterházy Károly Egyetem Programtervező informatikus szakára úgy, hogy semmilyen programozási tapasztalatom nem volt, ezért tudtam, hogy alapok nélkül nehezebb lesz ez az út. A döntés valójában abból következett, hogy már abban az időszakban is meg szerettem volna tanulni az alapokat ahoz, hogy robotikával, hardverekkel tudjak dolgozni, ezt az irányt találtam, amivel szívesen foglalkoznék a jövőben. Tudtam, hogy az egyetemen, ezen a szakon erre lesz lehetőségem.

Az egyetemi éveim alatt ezt az irányt csak erősíteni tudtam, mind lelkesedésben, mind tapasztalatokban. Emellett megismerkedtem a C++ nyelvvel is, amelyet furcsa módon sikerült is megkedvelnem, hiszen ezt a programozási nyelvet az egyetemen nem túl sokan választják, komplexitása miatt. Pedig szerintem, aki szereti a kihívásokat, egy elég sokoldalú nyelvet tudhat magának a C++ által.

Abban biztos voltam, hogy hardverekkel szeretnék dolgozni a szakdolgozatomban is, így kerültek az ötletek közé az okosotthon (smarthome), illetve az IoT témakörök. Viszont ezen a téren elég széles a lehetőségek köre, így elég tanácsokkal voltam, hogy mit is válasszak. Egyik szaktársam pont ezekben a témakörökben írta a szakdolgozatát, egy beléptető rendszer webszolgáltató részét, amelyhez hardveres megvalósítást is szánt. Ezért ötletként felvetette, hogy ha szeretném, akkor azt a részt elkészíthetném.

Valójában először még nem sejtettem, hogy ez lesz az, amit szakdolgozatként meg fogok valósítani, de ahogy egyre több részletet beszéltünk meg vele kapcsolatban, ez volt az, ami a legjobban felkeltette az érdeklődésemet.

A beléptető rendszer megvalósítása előtt és közben felmerülő kérdések hatására úgy döntöttem, hogy nem egy, hanem két változatot szeretnék elkészíteni (Teljesen elosztott, Csillagpontos megoldás), amelyeket a végén össze tudok hasonlítani, amely tanulságos lehet.

A dolgozatom végül ennek a két változatnak a bemutatásáról, majd pedig összehasonlításáról szól. Az eszközökre írt programkód C++-ban íródott, amely az egyik legelőnyösebb nyelv az IoT eszközök fejlesztésének világában.

Az egész projekt megtalálható Github-on, a következő címen:
<https://github.com/jalexandra/yii8yw-thesis>.

1. fejezet

A rendszer bemutatása

1.1. Internet of Things (IoT)

Az Internet of Things olyan készülékeknek, eszközöknek, berendezéseknek és gépeknek a hálózata, amelyek különböző szenzorok segítségével képesek a fizikai világról adatokat gyűjteni, majd pedig ezeket az adatokat az interneten keresztül továbbítani más eszközöknek.

A legfőbb feladata az olyan adatok gyűjtése, amelyeket intelligens eszközök nélkül nagyon nehéz lenne megszerezni.

Az IoT az elmúlt években robbanásszerűen, az egyik legjelentősebb technológiává vált. Ennek köszönhetjük, hogy egyre több elérhető okos eszköz létezik, amelyek képesek interneten keresztül kapcsolatba lépni egymással és adatokat küldeni és fogadni egymás között. Ennek gyors fejlődésnek több oka is, van, de ezek közül a legjelentősebbek, az eszközök és a szenzorok ár csökkenése (az újabb és újabb gyártási technológiáknak köszönhetően), és az új hálózati protokollok (pl. MQTT) megjelenése.

Az IoT segítségével az eszközeink interakcióba léphetnek egymással, és a saját feldatataikat egymáshoz, és a környezethez igazíthatják. Vegyük például egy okoshőmérőt, ami érzékeli, hogy a lakásban egy beállított hőfoknál melegebb van. Erre bekapcsolhat egy klíma rendszer, hogy lehűtse a helyiséget, de akár az is lehet, hogy egy kinti hőmérő alapján érdemesebb szellőztetni, mint klímát kapcsolni. Ezek az eszközök mindenfolyamatosan kommunikálnak egymással, és emberi beavatkozás nélkül tudják végezni a feladatukat.

Létezik IIoT (Industrial IoT), amely az IoT ipari környezetben vett megfelelője. Ennek a legfőbb felhasználási területei pl. az intelligens gyártás, okos város, okos energiahálózat stb.

1.2. Webservice

A beléptető rendszerem Dombi Tibor Dávid szakdolgozatának a folytatása, az általa írt webservice-t, valamint az admin felületét használja.

Ahogy Tibor is írta, a piacra számtalan hasonló rendszert találni, de ezeknek egyike sem igazán testre szabható. A legtipikusabb változatok csupán kevés (sok esetben csak egyetlen) zárat tudnak vezérelni, és ezekben egyáltalán nem, vagy csak korlátozottan van lehetőségünk szabályok felvitelére. [2]

A „komolyabb” rendszerek rendkívül drágák, és legnagyobb részük csak szolgáltatás formájában igényelhető, amit általában csak a szolgáltató emberei tudnak beüzemelni.

Ennek a rendszernek soha sem az volt a célja, hogy egy „átlag” ember, mindenféle programozói tudás nélkül be tudja üzemelni, hanem az, hogy más fejlesztőknek egy könnyen hozzáférhető kiindulási alapot adjon egy, a saját elvárásainknak teljes mértékben megfelelő, ugyanakkor egyszerűen használható rendszer elkészítésének.

Az eredeti projekt rendkívül nagy hangsúlyt fektetett arra, hogy a lehető legolcsóbban megvalósítható legyen a rendszer, így ezt én is szem előtt tartottam a saját munkám során. Fontos volt az is Tibor dolgozatában, hogy a jövőben ezzel a rendszerrel dolgozó fejlesztőknek minél könnyebb dolga legyen, ezért amit csak lehetett dokumentált, illetve publikusan elérhetővé tette a projektet, ezekkel a dokumentációkkal együtt. [2] A példáját követve, én is feltöltöttem az összes kódot, tervet és leírást a publikus GitHub repository-mba.

A Tibor által készített webservice Laravel 9-re íródott és készült hozzá egy Angular admin felület is.

A webservice biztosít egy Rest API-t, amelyen keresztül a kommunikáció történik. Az autorizációhoz JWT-t (1.3) használ. Jelenleg a beléptetéshez nem szükséges JWT tokent küldeni, mert ehhez a kéréshez nincs azonosítás, viszont ez a jövőben valószínűleg változni fog. Ha valódi környezetben szeretnénk a rendszert használni mindenképpen le kellene védeni ezt a végpontot is. [2]

1.2.1. Beüzemelés

Tibor a project „*README*”-jében részletezi a projekt rendszerkövetelményét, illetve az telepítést lépésről lépésre, így itt csak a saját tapasztalataimról írok.

A projekt mellékel egy kis TypeScript fájlt, ami „telepítő varázslóként” működik és a `yarn wizard` parancssal tudjuk elindítani. A lényege az lenne, hogy kérdéseket tesz fel (pl. adatbázis típusa, kapcsolódási adatok stb.), és azok alapján, amiket választottunk elkészíti a konfigurációs fájlokat, amiket majd a projekt használni fog, illetve összeállít nekünk egy parancs sorozatot, amit lefuttatva minden telepít és be is állít nekünk. [3] Sajnos a projekt inaktív volt az elmúlt pár hónapban így a script nem volt kompatibilis a legújabb Node.js verzióval. Ez a probléma egyelőre nem lett megoldva, így jelenleg

csak a manuális telepítés lehetséges.

Első nekifutásra Docker-rel szerettem volna használni a projektet, de így a webszervice nem volt elérhető a helyi hálózatomról, csak a `localhost`-on. Ezt a hibát eleinte nem tudtuk megoldani, ezért inkább *natívan* konfiguráltam, és így használtam a webszert a továbbiakban. Ennek a megoldása, mint később kiderült, egyszerűen csak a *Laravel Sail* package frissítése volt.

Ezután nekiláthattam a két záram felvitelének az admin felületre (egy-egy a két implementációnak), két teszt dolgozó felvitelének (két RFID kulestartó állt a rendelkezésemre). A zárakat csoportokba, a dolgozókat pedig csapatokban kellett rendeznem, ezután a csoportokat és a csapatokat köthettem össze szabályokkal. A tesztelésre elég volt egyetlen egyszerű szabály, ami szerint az egyik RFID-t bármikor, a két zár közül bármelyiken átengedheti a rendszer.

Itt derült fény két problémára: Az első, hogy az átengedést kezelő végpont semmilyen módon nincs authentikálva, tehát bárki, bárhonnan küldhet oda üzeneteket. A második az volt, hogy a dokumentáció szerint az eszköz azonosítója (`device_id`) bármi lehet, ami egyedileg azonosítja azt a zárat, tehát lehetne például a mikrokontroller saját azonosítója (ESP esetén: `ESP.getChipId()`[4]), ez azonban hibásan működik, és csak azt az azonosítót fogadja el, amit az adatbázisban ID-nek kapott. Ez egy *Auto Increment* mező. Erre a két problémára sem készült megoldás a webszertaban egyelőre.

1.3. Azonosítás

Az azonosításhoz (authentication) JWT-t (JSON Web Token) használ a rendszer, amely egy kompakt és biztonságos módja az adatok átvitelének. A JWT tokenek egyszerre aláírtak és titkosítottak is.

3 részből áll: fejlécből, *payload*-ból és az aláírásból. A fejléc magáról a tokenről tartalmaz információt pl. a titkosítás algoritmusa. A *payload* maga az üzenet pl. a user adatait, illetve a token érvényességét tartalmazza. A harmadik rész pedig a szerver digitális aláírása. Az egész token *base64url* kódolást használ, ami annyiban különbözik a hagyományos *base64*-től, hogy csak olyan karaktereket tartalmaz, amik használhatóak URL-ben. Ezek az angol abc kis- és nagybetűi, számok, kötőjel (-) és az aláhúzás jel (_). [5]

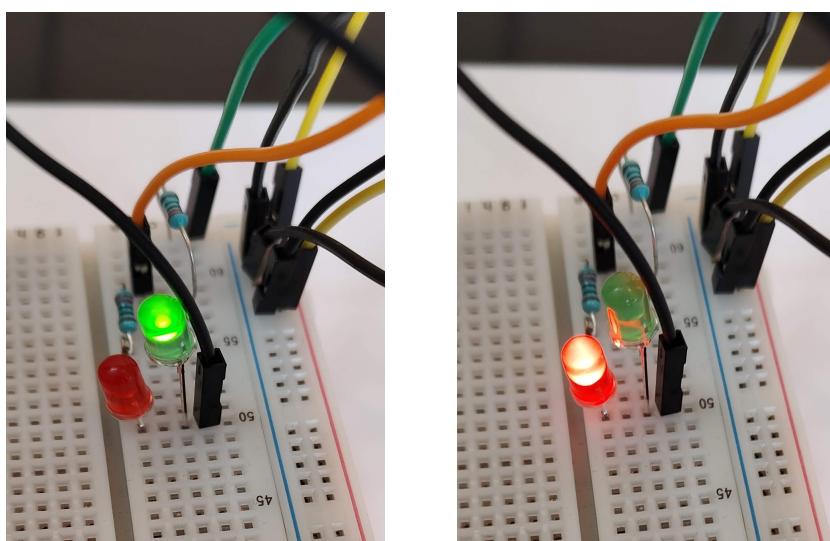
Ezt a JSON web tokent bejelentkezéskor elküldi a szerver és a kliens ezt tárolja, ezután minden üzenetéhez csatolnia kell a fejlécben úgy, hogy az *authorization* értéke a **bearer** szó, egy szóköz, majd a kapott token.

Mindez azért szerepel a dolgozatban, mert a jövőben a backend a beléptetéshez használt végponton is fog használni JWT-t.

2. fejezet

Teljesen elosztott megoldás

A teljesen elosztott megoldás lényege, hogy minden ESP közvetlenül tud kommunikálni a Rest API-on keresztül a webservice-al. Az ESP-k wifi kapcsolaton keresztül továbbítják az NFC olvasótól kapott információt a REST API felé, majd a válaszban kapott információ alapján jelet küldenek a zárnak és kinyitják, vagy zárva tartják azt. A user-t tájékoztatja egy piros vagy zöld led arról, hogy a zár melyik állapotba került.



(a) Zöld led

(b) Piros led

2.1. ábra. Ledes visszajelzés

2.1. Technológiák

2.1.1. RFID

Az RFID (Radio Frequency Identification) egy vezeték nélküli kommunikációs technológia melynek számtalan felhasználása létezik, mint az érintés mentes fizetés, az állatok

azonosítása, a különböző tárgyak nyomon követése. A szakdolgozatom szempontjából legjelentősebb az emberek azonosítása, illetve a belépési jogosultságuk ellenőrzése. [7]

A rendszer két fő elemből áll: Egy tag-ból, amin az azonosító tárolódik, és egy olvasóból, ami képes „lekérni” ezt az azonosítót a tag-től.

Egy RFID tag három részből áll: egy antennából, egy integrált áramkörből, ami tárolja és feldolgozza az információt, és magából a tárolóból (pl.: a műanyag ház, amiben benne van). A tag-ek különböző méretben és formában kaphatók, így a felhasználásuk is széles körben elterjedt. [6] Három fő típusuk a **passzív olvasó aktív tag (PRAT)**, az **aktív olvasó passzív tag (ARPT)**, és az **aktív olvasó aktív tag (ARAT)** utóbbi használhat **fél-passzív (BAP)** tag-eket is. [7]

Az **aktív tag**-ek saját áramforrást használnak, ezért nagyobbak, mint a passzív tag-ek. A saját azonosítójukat folyamatosan sugározzák.

A **passzív tag**-ek az olvasó által indukált áramot használják a működéshez, ezért jóval kisebbek, mint az aktív tag-ek, és csak „válaszolnak” az olvasó által küldött kéresekre. Ezt a kérést *interrogation signal*-nak szokás nevezni. Rendszerint ezek a tag-ek csak kis távolságról működnek.

A **fél-passzív** (battery-assisted passive - BAP) **tag**-ek egy kis akkumulátorral rendelkeznek, amelynek elsődleges szerepe, hogy segítse a jel erősítését. A BAP tag-ek átmenetet képeznek a két típus között.

A tag-ek különböző frekvenciákon, és különböző jellemzőkkel működhetnek. Ezek a **LF** (Low Frequency), **HF** (High Frequency), **UHF** (Ultra High Frequency), és a **SHF** (Super High Frequency) technológiák. Továbbá, minden tag lehet csak olvasható (**RO**), amit a gyárban programoznak fel. Egyszer írható, sokszor olvasható (**WORM**), vagy sokszor írható, sokszor olvasható (**RW**). [8]

Azért ezt a technológiát választottam a szakdolgozatom megvalósításához, mert olcsó az olvasó és hozzá a tag-ek is. Illetve rendkívül egyszerű, sokoldalú az eszköz és a használata is, mind programozás, mind felhasználás szempontjából. Az általam használt RFID egy ARPT (aktív olvasó passzív tag), és egy csak olvasható alacsony frekvenciás tag-eket használó rendszer.

Az **RDM6300** RFID olvasó modult használtam, amely 125 KHz (LF) frekvencián olvas. Ezt az RFID kártyánál is figyelembe kell venni, más frekvenciájú kártyát nem fog tudni olvasni. Fontos, hogy az olvasó 5 Voltos áramforrást igényel, ezért egyes mikrokontrollerek esetén, amelyek csak 3,3 Volt-ot tudnak biztosítani, egy külön áramforrásról kell majd ellátnunk.[9]

2.1.2. L293 D

A zárak mozgatásának szemléltetésére egy DC motort használtam, amelyet egy L293 D motorvezérlő segítségével irányítottam. Mivel az ESP nem elég erős ahhoz, hogy

a motort közvetlenül hozzá kapcsoljuk, ezért a motor működtetéséhez szükség lenne egy tranzisztorra. Ennek a megoldásnak a hiányossága, hogy így csak az egyik irányba tudjuk forgatni a motort. Természetesen a DC motor helyett lehetne mászt is használni pl. szervomotor, elektromotor, elektromágneses zár stb.

Az L293 D egy integrált áramkör (Integrated Circuit - IC), ami négy fél-H hídból áll, így képes két DC (vagy egyetlen *stepper*) motor vezérlésére egymástól függetlenül. Segítségével a motor minden irányba forgatható attól függően, hogy mely pinekre kapcsolunk áramot. [10] Ez rendkívül fontos, ha egy zárat szeretnénk kinyitni, majd pedig vissza zárni.

Arduino-hoz elérhető egy *Arduino Motor Shield* nevű eszköz, amelynek a lényege ugyanaz [11], mint az általam használt L293D-nek, viszont sokkal nagyobb és drágább, ezért mindenki a megoldásnál maradtam a motorvezérlő használatánál.

2.1.3. ESP8266

Az ESP8266 egy Espressif Systems által gyártott, aránylag olcsó, 32 bit-es, WiFi-s mikrochip, amely kifejezetten IoT projektekre lett kifejlesztve. [12]

Az **ESP8266-01** egy 8 pines, 3,3 Volton működő mikrokontroller, amely képes WiFi-re csatlakozni. Tartalmaz egy firmware-t, amelynek segítségével AT parancsokkal tudunk csatlakozni a hálózatra. Többféleképpen is lehet programozni pl. Arduino-val vagy programozóval. Ez a legolcsóbb, legkisebb áramfelvételű eszköz, ami WiFi-re is tud csatlakozni, és megvalósítható lehet vele a rendszer működése. Én nem ezt használtam, de ez egy olcsóbb megvalósítása lehet a rendszernek.

Az **ESP8266 NodeMCU** már inkább egy Arduino UNO-hoz (lásd 3.1.4) hasonlító eszköz, viszont a mérete sokkal kisebb. Ez a mikrokontroller is 30 pines, illetve 5 Volton működik. Az Arduino-val ellentétben az eszköz hátoldalán találhatóak a pinek, így ezt könnyebben lehet próbápanellel használni. Az Arduino UNO-val ellentétben és az ESP8266-01-hez hasonlóan, a NodeMCU képes WiFi-re csatlakozni. Én ezt az eszközt használtam a beléptető rendszer teljesen elosztott megoldásának megvalósítására, mert úgy láttam, hogy a jövőbeli projektjeimhez hasznos lehet a megismerése.

Az eszköz programozásához szükség van egy **CH340** vagy egy **CP2120** driver-re, ez, a használt ESP-től függ, amely lehetővé teszi a soros kommunikációt a számítógép USB portja és a mikrokontroller soros portjai között.[13]

Az **Arduino IDE** segítségével egyszerűen lehet programozni az eszközt, csupán installálni kell az ESP8266 by ESP8266 Community-t, amely tartalmazza az ESP8266 chip-el rendelkező board-okat, ezután már csak ki kell választani a megfelelő eszközt.[14] Majd az Arduino IDE-ben a Tools-on belül személyre tudjuk szabni a beállításokat.

Az ESP sikeres programozásához a Tools-on belül a Flash Mode-ot DIO-ra kell

állítani. A DIO (Dual I/O) egy SPI (Serial Peripheral Interface) flash mód, az olvasás és írás két pinen keresztül történik (MOSI, MISO), ezáltal kétszeres adatsebességgel (2 bit/órajel).[15] Fontos, hogy az ESP RX, TX pinjeit szabadon kell hagyni programozáskor, tehát, ha USB-n keresztül programozzuk az eszközt, ezeknek a lábaknak szabadon kell maradniuk, különben az Arduino IDE nem fogja tudni elérni az ESP-t és a kód feltöltése sikertelen lesz.

Az ESP **csak 2,4 GHz-es hálózatra** tud felcsatlakozni, ezért kialakítottam egy külön hálózatot a projekt számára egy régebből megmaradt 2,4 GHz-es *TP-link* router segítségével. Ez azért kellett, mert az 5 GHz-et és 2,4 GHz-et is támogató fő routerre nehezebben tudott csatlakozni.

Vezetékek színeinek jelentése a bekötésben:

Általánosan:

- **fekete**: 0V, GND
- **piros**: 5V-os áramforrás
- **citromsárga**: 3,3V-os áramforrás

RFID olvasó esetében:

- **narancssárga**: TX pin
- **zöld**: RX pin

Led-ek esetében:

- **narancssárga**: piros led áramforrása
- **zöld**: zöld led áramforrása

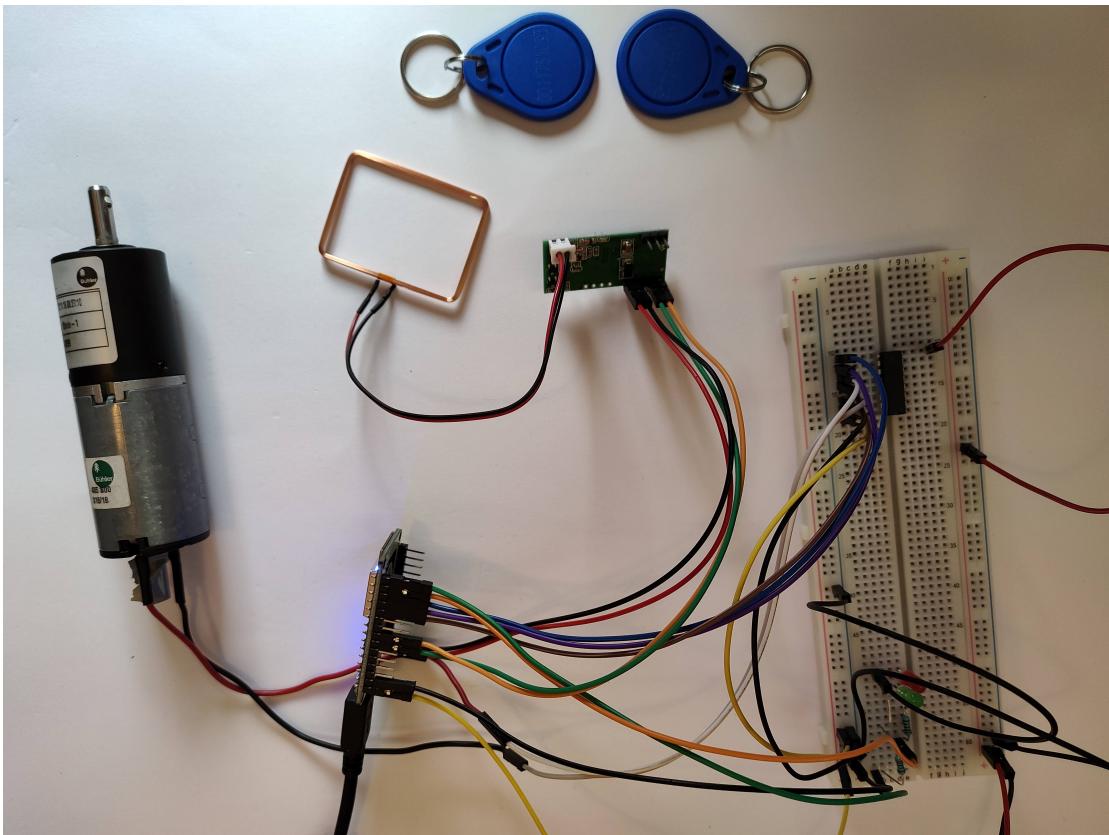
Motorvezérlő lábai:

- **barna**: Input 4 pin
- **szürke**: Output 4 pin
- **fehér**: Output 3 pin
- **lila**: Input 3 pin
- **sötét kék**: Enable 3, 4 pin

Ennek a megoldásnak a **kapcsolási rajza** a következő címen tekinthető meg, mivel így könnyebben megtekinthető, nagyítható:

https://raw.githubusercontent.com/jalexandra/yii8yw-thesis/main/doc/images/distributed_bp.png

Az alábbi (2.2) képen látható a rendszer próbapanelben összerakott prototípusa.



2.2. ábra. A teljesen elosztott megoldás próbapanelben

2.2. RFID elérés

Az RFID olvasó vezetékekkel kapcsolódik az ESP D5, D6 pin-jeihez, egy 5V-os tápegységhez, illetve a GND-hez. Az RFID olvasó, olvasó fejéhez közelítve az RFID kulcs-tartót/kártyát, továbbítja a beolvast adatokat az ESP-nek.

Az eszköz használatához szükség van a `SoftwareSerial` könyvtárra. Amely szoftveresen lehetővé teszi, hogy a mikrokontroller digitális pin-jein is folytathassunk soros kommunikációt. Egyszerre több ilyen port is lehet, viszont ezek nem tudnak ugyanabban az időben információt fogadni. Egy port pedig nem tud egyszerre adatokat fogadni és küldeni. [16]

A `SoftverSerial` osztály `constructor`-ának hívásakor meg kell adnunk, hogy az olvasó RX, TX lába, mely ESP-beli lábakhoz csatlakozik. Majd a `setup`-ban be kell állítanunk a bitrátát (*baud rate*) 9600-ra, az `RFID.begin()` segítségével. A működést a Serial Monitor-ral számítógépünkéről ellenőrizhetjük. Amennyiben a Serial Monitor és a mikrokontroller eltérő frekvenciára van állítva, a Serial Monitoron olvashatatlan karakter sorozat jelenik meg vagy nem is jelenik meg semmi. Fontos, hogy a hár-

térben a `SoftwareSerial` is a hardveres soros portra továbbítja az adatot, ezért a `Serial.begin()`-el azt is inicializálnunk kell, még akkor is, ha egyébként nem használnánk az RX, TX portokat, vagy az USB-t. Az `RFID.read()` beolvassa a következő elérhető byte-ot a tag-en. Az `RFID.available()` visszaadja, hogy még hány olvasható byte van a tag-en.

```

1 #include <SoftwareSerial.h>
2
3 SoftwareSerial RFID(RX, TX);
4
5 void setup(){
6     Serial.begin(9600);
7     RFID.begin(9600);
8 }
9
10 void loop() {
11     while (RFID.available() > 0) {
12         delay(5);
13         Serial.println(RFID.read());
14     }
15 }
```

2.1. kód. RFID olvasás példa

2.3. Rest elérés

A Rest API eléréséhez az ESP-nek fel kell csatlakoznia a megfelelő hálózatra, amelyhez szükség van az **ESP8266WiFi** könyvtárra, amelyet csak be kell importálnunk a kódunkba az *include* kulcsszó segítségével. A `WiFi.disconnect()`-el le tudunk csatlakozni a hálózatról, ezzel garantálva, hogy az ESP nincs hálózatra csatlakozva. A `WiFi.begin()`-el, meg kell adnunk az SSID-t, illetve a jelszót, ha van, ezután már fel tudunk csatlakozni a megadottaknak megfelelő hálózatra. [17]

Ha a webservice-t DEVELOP módban, a `php artisan` parancssal futtatjuk, akkor adjuk meg a `--host=0.0.0.0` paramétert, hogy a localhost-on kívülről is elérhető legyen. Amennyiben a webservice más hálózaton fut még egyéb konfigurációra is szükségünk lehet.

```

1 #include <ESP8266WiFi.h>
2
3 #define WIFI_SSID "SSID"
4 #define WIFI_PASS "password"
5
6 void setup(){
7     WiFi.disconnect();
8     WiFi.begin(WIFI_SSID, WIFI_PASS);
```

```

9
10    int attempt = 0;
11    while (WiFi.status() != WL_CONNECTED) {
12        delay(3000);
13        Serial.print("Connecting. Attempt: ");
14        Serial.println(++attempt);
15    }
16 }
```

2.2. kód. Wifi-re kapcsolódás példa

Szükség lesz az **ESP8266HTTPClient** könyvtárra, amelyet szintén be kell importálnunk az *include* kulcsszó segítségével. A HTTPClient osztály segítségével tudjuk elérni, majd pedig http metódusokon keresztül kommunikálni a megfelelő webservice-al. Példányosítanunk kell az osztályt, hogy elérjük a szükséges metódusokat.

A `http.begin()` metódusnak meg kell adnunk egy client-et, majd pedig az elérési utat, hogy hol tudja elérni a service-t. Ezután `http.GET()` elküld egy GET kérést, amelyre választ is kap, viszont itt más http metódus is szerepelhet. A `http.end()` pedig lezárja a kapcsolatot.

Ha sikerült felcsatlakozni a WiFi-re, akkor a következő kód sorokat kell hozzáadnunk az előzőhöz, hogy elérjük a megfelelő webszervert:

```

1 #include <ESP8266HTTPClient.h>
2
3 void setup(){
4     // A setup nem változik
5 }
6
7 void loop(){
8     if (WiFi.status() == WL_CONNECTED) {
9
10         HTTPClient http;
11         WiFiClient client;
12
13         http.begin(client, "http://192.168.0.100:3000/");
14         int httpCode = http.GET();
15
16         if (httpCode > 0) {
17             String payload = http.getString();
18             Serial.println(payload);
19         }
20         http.end();
21     }
22     delay(3000);
23 }
```

2.3. kód. HTTPClient kódok

Fontos, hogy az adatokat JSON formátumban küldjük el. Alapértelmezetten a HTTPClient `text/plain` formátumban küldi el az adatokat, viszont az általam használt Rest API csak `application/json` formátumot fogad el. Ha a következő sor nem szerepel a kódban, akkor hibát fog visszaadni a backend.

```
1 http.addHeader("Content-Type", "application/json");
```

A backend-nek el kell küldenünk, PUT metódus segítségével egy `lock_id`-t, amely a zár azonosítója, illetve a `worker_rfid`-t, amit beolvastunk az RFID olvasóval. Annak érdekében, hogy meg tudjam nézni a kérésre érkező választ, elmentettem azt, egy `int` változóba.

```
1 int httpCode = http.PUT(body);
```

2.4. Zárak vezérlése

Mielőtt bármelyik pin-t használnánk, először be kell állítani a `pinMode()` segítségével, hogy *input*-ként vagy *output*-ként szeretnénk használni. Ezt általában a `setup` részben szokás beállítani minden általunk használt pinre.

A motor forgatását egy `motor` elnevezésű függvény irányítja, amely paraméterként vár egy `integer`-t, jelenleg 0-tól 2-ig. Ez alapján balra vagy jobbra kezdi el forgatni a motort, vagy pedig megállítja azt.

```
1 void motor(int direction){  
2     switch(direction){  
3         case 0:  
4             digitalWrite(MOTOR_E, LOW);  
5             digitalWrite(MOTOR_1, LOW);  
6             digitalWrite(MOTOR_2, LOW);  
7             break;  
8         case 1:  
9             digitalWrite(MOTOR_E, HIGH);  
10            digitalWrite(MOTOR_1, HIGH);  
11            digitalWrite(MOTOR_2, LOW);  
12            break;  
13        case 2:  
14            digitalWrite(MOTOR_E, HIGH);  
15            digitalWrite(MOTOR_1, LOW);  
16            digitalWrite(MOTOR_2, HIGH);  
17            break;  
18    }
```

2.4. kód. Motor vezérlés

Először megvizsgálom a webservice-tól érkező státusz kódöt, majd ez alapján meg-hívom a függvényt, a megfelelő paraméterrel, itt `delay` segítségével állítom be azt,

hogy mennyi ideig forgassa a motort, vagy éppen állítsa meg. Azért a `delay` függvényt használtam, mert ilyenkor ez a függvény blokkol bármely más tevékenységet, ez itt szükséges, hiszen amíg a zárat vezéreljük, nem szeretnénk mással foglalkozni.

```
1  if(status == 202){  
2      digitalWrite(GREEN_LED, HIGH);  
3      motor(RIGHT);  
4      delay(MOTOR_MOVE);  
5      motor(STOP);  
6      delay(OPEN_TIME);  
7      motor(LEFT);  
8      delay(MOTOR_MOVE);  
9      motor(STOP);  
10     digitalWrite(GREEN_LED, LOW);  
11 }
```

2.5. kód. motor függvény hívások

Az előző kódrészlet a motor függvény hívásán kívül, azt is szemlélteti, hogy hogyan történik a dolgozó beengedése a zárral ellátott részlegbe. Ha 202-es státuszkódot kapott vissza az ESP, akkor felvillan egy zöld led, amely visszajelzésként szolgál az adott dolgozónak, hogy hamarosan nyílik a zár. A zár kinyitása után megállítjuk a motort, hogy legyen ideje belépni a dolgozónak az adott ajtón, majd pedig visszazárjuk a zárat és lekapcsoljuk a ledet.

400-as vagy annál nagyobb státuszkód akkor fordulhat elő, ha az olvasás során hibás adatot olvastunk, az olvasott RFID nincs az adatbázisban, ha a dolgozó nem léphet be az adott ajtón, továbbá, ha backend hiba történik (500-as státuszkód). Ekkor egy piros led villan fel, jelezve, hogy az ajtó zárva marad.

```
1  else if(status > 399){  
2      digitalWrite(RED_LED, HIGH);  
3      delay(WAIT_LED);  
4      digitalWrite(RED_LED, LOW);  
5 }
```

2.5. Tesztelés

A dolgozat megvalósításához az Arduino IDE-t használtam, mivel ezzel a legegyszebb a fejlesztés. Limitáltsága viszont, hogy nem igazán vannak ehhez a környezethöz jó teszt keretrendszerök. Léteznek olyanok, mint például az ArduinoUnit és az AUnit, de ezek mind a mikrokontrolleren futnak, miközben a valós kódot is lefuttatják ott, így inkább nevezhetők *debug library*-knek, mint teszt *framework*-öknek. [18] A *PlatformIO* környezetben valamivel alacsonyabb szinten kell dolgoznunk, de telepíthető hozzá egy *Test Runner* modul, ami támogatja, hogy *Unit Test*-eket írunk és azokat a szá-

mítógépünkön futtassuk, anélkül, hogy egyáltalán szükségünk lenne a fizikai eszközre.
[19]

Én csak manuálisan teszteltem a kódot, a rész eredményeket pedig egyszerűen a *Serial Monitor*-ra kiíratva figyeltem. Ennek a manuális tesztelésnek talán egy jobb módja, ha a kódunkat először valamilyen szimulátorban próbáljuk ki. Az egyik legkomplexebb ilyen rendszer a wokwi.com. Lehetőségünk van többféle mikrokontroller-t használni, és számtalan elektronikai alkatrész áll a rendelkezésünkre.

Legelőször az RFID olvasást próbáltam ki, hasonló kóddal, mint az RFID elérésnél található példakód. Ha már a beolvasás sikeresen ment, következett a Wifi-re csatlakozás, amely szintén hasonló programkóddal történt, mint a Wifi-re kapcsolódás példa-kód. Ezután következhetett a szerver elérése, majd a HTTP kérések küldésének tesztelése. Először csak egy kisebb Teszt szervert próbáltam elérni, egy GET kéréssel. A szerver csak annyit csinált, hogy jelezte a csatlakozást, majd pedig kiírta a *request body*-t. Miután itt is megoldódtak a problémák, akkor az RFID olvasó által beolvasott adatot küldtem el, ugyanennek a szervernek egy PUT metódussal, a szerver pedig kiírta a küldött azonosítót, így ellenőrizhettem annak formátumát. Majd csak ezután írtam meg a jelenlegi kódváltozatot, amely már a rendes *backend*-hez kapcsolódik és annak küldi el az adatokat, a megfelelő formátumban. Persze eleinte ezzel is akadtak problémák, amelyeket a különböző részteredmények *Serial*-ra való kiíratásával, a *Serial Monitoron* figyeltem.

2.6. Lehetséges problémák

A teljesen elosztott megoldás egyik legfőbb problémája, hogy jelenleg nem tudjuk követni azt, hogy egy eszköz elérhető-e. Ezért hiba esetén az üzemeltető nem kap semmilyen értesítést a meghibásodott eszkösről addig, amíg egy user észre nem veszi azt. Továbbá a hiba okát is nehézkes kideríteni, hiszen jelenleg a *debug*-olásnak egyetlen módja, ha számítógépre kötjük az eszközt, és figyeljük a *Serial Monitor*-ba érkező üzeneteket. Erre a problémára a továbbfejlesztési lehetőségek között (lásd 5. fejezet) található egy lehetséges megoldás.

Egy másik főbb hiba lehetőség, egy stabil 2,4 GHz-es hálózatot kell biztosítani az ESP-knek. Ez természetesen kisebb épület esetén nem olyan nehéz feladat, viszont pl. egy több emeletes irodaház vagy egy ipari épület esetén már felmerülhetnek problémák. A hálózati hibák kikerülése érdekében érdemes megkeresni azokat az eszközöket, épületi elemeket, amelyek a hálózat stabilitását veszélyeztetik és ezek tudatában a legoptimálisabban kiépíteni azt.

3. fejezet

Csillagpontos megoldás

A csillagpontos megoldás egy alternatív implementáció. Lényege, hogy csak egy központi eszköz, például egy Raspberry PI kommunikál közvetlenül a Rest API-al, így a mikrokontrollerek csak az RFID olvasóval, a zárakkal, illetve a Raspberry-vel folytatnak kommunikációt.

3.1. Technológiák

3.1.1. SSH

Az SSH (Secure Shell) egy olyan hálózati protokoll, amit arra terveztek, hogy különböző távoli szolgáltatásokat használjunk biztonságosan (pl. *remote login*, *remote command execution*), egy nem titkosított hálózaton keresztül. Ezt aszimmetrikus titkosítással éri el. [27]

Minden kliens generál egy kulcs párt (pl. `ssh-keygen` parancs), ami egy publikus és egy privát kulcsból áll. A privát kulcs a kliens birtokában marad, még a publikus kulcsot el kell küldenünk a szerverre. Ennek a kulcsnak a birtokában bárki tud üzenetet titkosítani, de az csak a privát kulccsal fejthető vissza. Amennyiben a szerver megbízhatónak találja ezt a publikus kulcsot eltárolja azt az `authorized_keys` fájlban. [28][29]

A csatlakozáshoz a `ssh USERNAME@RASPBERRY-IP` parancsot használhatjuk, ahol a `USERNAME` a bejelentkezéshez használt user neve (alapból *pi*) és a `RASPBERRY-IP` a Raspberry által használt IP cím. Ha jelszóval szeretnénk bejelentkezni, akkor a kulcs generálás és feltöltés automatikus, de a jelszót minden bejelentkezésnél meg kell adnunk újra.

Biztonsági okokból érdemes feltölteni a saját publikus kulcsunkat (pl. `ssh-copy-id` parancs) és a jelszavas bejelentkezést minél korábban letiltani. Tovább az alap 22-es portot érdemes valami másra cserélni (pl. 2222).

3.1.2. Raspberry PI

A szakdolgozatomban a Raspberry PI 4 model B-t használtam (2GB RAM-os változat). A jelenlegi (2022-ben próbáltam beszerezni az eszközt) globális ellátási problémák, és a chip hiány miatt az alaplap nem volt önmagában elérhető. Ezért alternatív megoldás lehet a Raspberry PI 4 helyettesítésére, a Banana Pi, az Orange PI, Asus Tinker Board, Beaglebone stb. hazánkban elérhető alaplapjai. [25] Ezeket az eszközöket szokás Single Board Computer-eknek (SBC) nevezni is.

Először tartottam tőle, hogy nehéz lesz boldogulnom a Raspberry PI-vel, hiszen még nem dolgoztam vele ezelőtt. Nem tudtam, hogy mennyire lesz bonyolult megoldanom azt, hogy elérjem a kívánt működést. Viszont amint elkezdtem vele a munkát, rájöttem, hogy egész egyszerű vele dolgozni.

A Raspberry PI-nek, az ESP-hez és az Arduino-hoz hasonlóan legalább 30 pin-je van, 5 Voltból működik és van USB csatlakozója. Viszont az ESP-vel és az Arduino-val ellentében, szükséges hozzá egy USB-C-s töltő. A Raspberry 4 hajlamos melegedni, ezért oda kell figyelni rá, mindenkorban érdemes egy hűtőbordát tenni rá. Szükségünk lesz még egy Micro SD kártyára, amelyen az operációs rendszer és a szükséges programok, adatok tárolódnak. Sokkal többféle és darabszámmal csatlakozóval rendelkezik, mint az említett mikrokontrollerek, amelyeknek köszönhetően pl. kamerát, képernyőt vagy monitort is lehet hozzá csatlakoztatni.

A Rasprberry PI-nek szüksége van egy **operációs rendszerre**, hogy működni tudjon, ugyanúgy, mint egy számítógépnek. A hivatalosan támogatott operációs rendszer a Raspberry PI OS vagy másnéven Raspbian, de bármilyen más operációs rendszerrel használható, ami támogatja az ARM processzor architektúrát, általában valamilyen Linux disztribúciót szoktak használni. Ha a Raspberry-t csak szerverként szeretnénk használni, akkor érdemes a *Raspberry PI OS Lite*-ot használni, mert az egy kisebb, GUI nélküli verzió. Egy egyszerű és gyors módja az operációs rendszer telepítésének a *Raspberry PI Imager*. Ehhez az operációs rendszert egy microSD kártyára kell telepíteni a megfelelő beállításokkal, pl. meg kell adnunk, hogy melyik Wifi hálózatra csatlakozhat, illetve elérhetővé tesszük-e az SSH-n történő elérést, és ezt a Raspberry PI kártya olvasójába helyezve, a boot-olási folyamat után már használható is az eszköz.[20]

Amennyiben nem szeretnénk monitort és perifériákat csatlakoztatni a Raspberry-hez, akkor mindenkorban pipáljuk be az `enable SSH` opciót. Ezzel engedélyezhetjük a Raspberry SSH szerverét. (lásd 3.1.1)

Ebben a megoldásban csak ez az eszköz kommunikál a Rest API-val, USB-n keresztül kommunikál az Arduino-val, ami az RFID olvasóhoz, a led-ekhez és a zárhoz van kötve. Amikor az RFID olvasótól adat érkezik, tehát valaki ki szeretné nyitni a zárat, akkor a dolgozó beolvasott azonosítóját, illetve a zár azonosítóját elküldi a Rest API-nak. Az üzenetre érkező válasz alapján pedig jelet küld az Arduino-nak, hogy

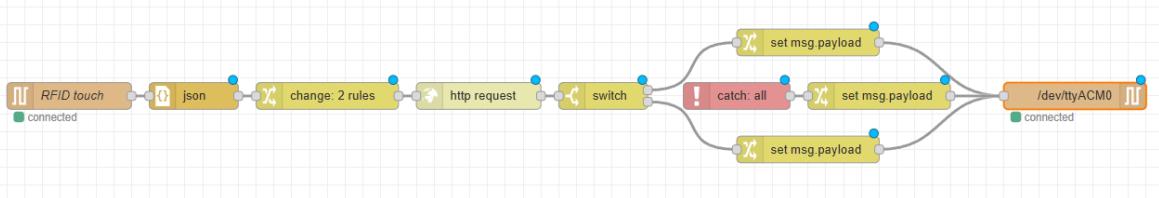
kinyithatja-e a zárat vagy zárva kell azt tartania.

3.1.3. Node-RED

A Node-RED egy szoftver, a hardverek, API-k és online szolgáltatások összekapcsolására. Ez az egyik legnépszerűbb IoT vezérlő megoldás. Egyszerűen installálható a Raspberry-re, és akár böngészőből a saját számítógépünkről is el tudjuk érni azt,[21] én is ezt a megoldást választottam.

Az egész rendszer JavaScript alapú és egy Node.js alapú webszerveren fut. A használata rendkívül egyszerű, egy vizuális programozási nyelvet használ (flow editor), amelyben minden általunk létrehozott flow egy JSON object-é fordul le. Ez a *JSON object* tartalmazza az összes *node*-ot, azok konfigurációit, illetve az ezek közti kapcsolatokat. Egy *flow* több „node-láncból” is állhat. A node-ok adatokat kapnak és küldenek egymásnak a saját feladatuknak megfelelően. Egyszerre több futó *flow* is lehet, ezek kommunikálhatnak is lehetnek egymással, de teljesen függetlenek is lehetnek. Úgy működnek, mint különböző, futó programok a rendszerben.[31]

Csak ki kell választani a program működéséhez szükséges *node*-okat, amik egy-egy utasításnak felelnek meg. Kétszeri kattintás hatására ezek szerkesztő felületén, amely minden *node* esetén különböző, meg kell adnunk a programunk számára megfelelő beállításokat, majd helyes sorrendben össze kell kötnünk őket. A *Deploy* gomb élesíti a változtatásokat, innentől a Raspberry-n fut az általunk írt program. A *debug node*-ok használata segíti a flow működésének a tesztelését, a *debug node*-ok eredményei a *debug message* fül alatt találhatóak. Amennyiben a számtalan elérhető *node*-ok közül egy sem felel meg a céljainknak, létrehozhatunk saját *node*-okat is *JavaScript*-ben.



3.1. ábra. Az általam használt flow

Általam használt *node*-ok:

- **serial in:** A *network node*-ok között található. A segítségével meg tudjuk mondaní, hogy melyik portról várjuk az adatot. Használat előtt ki kell választani a megfelelő portot. Ebben a megoldásban a Raspberry Pi vezetéken keresztül folytat kommunikációt egy Arduino UNO-val, amelytől a beolvasott adatokat kapja. Ahhoz, hogy a Node-RED segítségével *serial portok*-kal tudjunk kommunikálni, installálnunk kell a **node-red-node-serialport** NPM csomagot. [22]

- **json**: A *parser node*-ok között található, amint a kategória is mutatja, a beérkező adatokat alakítja át az Action listából kiválasztott módon. Ebben a megoldásban az *Always convert to JavaScript Object* lehetőséget használtam. Ez a *node* hibát dob, ha a konvertálás nem lehetséges.
- **change**: A *function node*-ok között található. Adott szabály vagy szabályok szerint változtat (Set, Change, Delete, Move) a beérkező adatokon. Arra használtam, hogy a kapott adatokból beállítsam a `device_id` és a `worker_rfid` értékét.
- **http request**: Szintén a *network node*-ok között található. Egy választott http metódussal kérést indít a megadott címre, a megadott adatokkal. Ezt a *node*-ot használtam, hogy egy *PUT* metódussal elküldjem a *Rest API*-on keresztül a beolvasott RFID-t. Hozzá kell adnunk a *Headers*-höz egy új sort, amely azt tartalmazza, hogy a *backend* milyen formátumban várja az érkező adatokat. Tehát az első mezőben az `Accept` szó, a másodikban pedig az `application/JSON` található.
- **switch**: Szintén a *function node*-ok között található. Ahogyan a neve is mutatja, úgy működik, mint más nyelvekben egy `switch-case` szerkezet. Arra használtam, hogy a *bservice*-tól kapott válasz alapján két irányba „ágaztassam” a flow-t. Az első ág az az eset, amikor a válaszban kapott JSON object *action* tulajdonságának értéke `"allow" (string)`. Ebben az esetben az Arduino-nak 1-est kell visszaküldenünk. minden egyéb esetben pedig 0-t. Ideértve azt is, ha hiba keletkezik.
- **catch**: A *common node*-ok között található. Segít, hogy az adott flow-ban törtenő esetleges hibákat kezelni tudjuk. Használatához ki kell választani, hogy az összes *node*-on, vagy csak a kiválasztottan szeretnénk kezelni a hibákat. Ez azért lényeges, mert így, ha az elküldött adat megsérül, az Arduino akkor is kap választ.
- **serial out**: A `serial in`-hez hasonlóan a *network node*-ok között található, viszont pontosan a fordítottja. Azt tudjuk megmondani vele, hogy melyik porton szeretnénk elküldeni a beérkező adatot. A segítségével küldöm el a választ az Arduino-nak, amely alapján majd a ledeket, illetve a zárat kell vezérelnie.

3.1.4. Arduino

Az Arduino UNO-t az egyetemen folytatott tanulmányaim során ismertem meg, emiatt már volt nálam egy, amivel akkoriban dolgoztam. Ez az eszköz család elég ismert, széles körben használják oktatási célokra sokszínűsége és egyszerűsége miatt. Ezért úgy gondoltam, hogy ebben a megvalósításban a Raspberry-vel ez az eszköz fog kommunálni, az RFID olvasót, a led-eket, illetve a motorvezérlőt ezzel az eszközzel szeretném

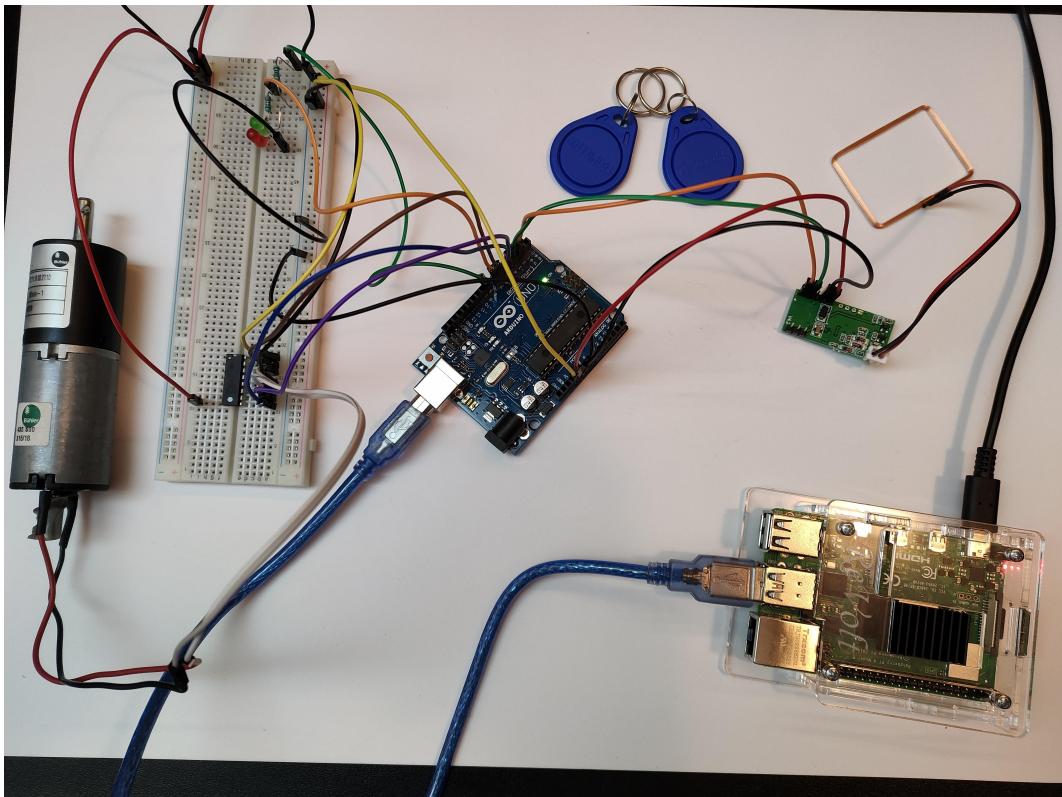
vezérelni. Itt kiemelném, hogy egy valódi rendszer kiépítés során inkább használnék valamilyen olcsóbb alternatívát, mint például az ESP-01. Ebben az esetben a soros portot kiváltanám WiFi kapcsolatra, ahol a Raspberry HTTP vagy MQTT szerverként működik.

Az Arduino UNO egy szintén 5 Volton működő, ATmega328P chip-el vezérelt, 6 darab analóg bemenettel, illetve 14 digitális pin-el rendelkezik, amelyek közül az utóbbi lehet Input és Output is. [30]

Ennél a megoldásnál törekedtem az ESP-nél leírt vezetékek színeinek a megtartására, így az, a leírás a csillagpontos megoldáshoz is alkalmazható.

Ennek a megoldásnak a **kapcsolási rajza** pedig a következő címen érhető el, mivel a dokumentumban kevésbé átlátható:

https://raw.githubusercontent.com/jalexandra/yii8yw-thesis/main/doc/images/centralized_bp.png



3.2. ábra. A csillagpontos megoldás

Az Arduino-hoz használt kód hasonló, az ESP-nél használthoz. Legalább is a led-ekkel és a motorvezérlővel történő kommunikáció esetében. Az általam használt Arduino UNO nem tud wifi-re csatlakozni, illetve nem kommunikál közvetlenül a webservice-al, így ennek az eszköznek a programkódjában csak a `SoftwareSerial` könyvtárra lesz szükségünk az RFID olvasó használatához.

Annak okán, hogy az Arduino `Serial`-on keresztül kommunikál az RFID olvasóval és a *Raspberry*-vel is, oda kell figyelni arra, hogy az RFID olvasóval csak addig folytas-

sunk kommunikációt, amíg az szükséges. Az Arduino egyszerre egy Serial portra tud figyelni, amelyet az RFID olvasó használata teljesen lefoglal, ezért, ha már beolvastuk a szükséges adatokat, meg kell szakítanunk az RFID olvasóval a kommunikációt ahhoz, hogy el tudjuk küldeni a Raspberry-nek az érkezett azonosítót, majd pedig fogadni tudjuk az arra érkező választ.

Az RFID beolvasása megegyezik az elosztott megoldásban használt kódokkal, viszont az azonosító elküldésére használt `send()` függvény már különbözik. A kapott `rfid` paraméter hosszát ugyanúgy vizsgáljuk, viszont itt nincs szükség a wifi kapcsolat ellenőrzésére. Ezért miután beolvastuk az azonosítót és összeállítottuk az elküldeni kívánt üzenetet, továbbítjuk azt a Raspberry felé.

```

1  String send(String rfid){
2      if(!rfid.length()){
3          return "";
4      }
5      String body = String("{\"deviceId\": \""
6          + DEVICE_ID
7          + "\", \"rfid\": \""
8          + text
9          + "\"}");
10     Serial.println(body);
11     return wait_for_response();
12 }
```

3.1. kód. `send()` függvény

A `send()` függvény meghívja a `wait_for_response()`-t, amely legelőször megszakítja az RFID olvasóval a kommunikációt, enélkül az olvasó lefoglalná a `Serial`-t, így nem kapnánk meg a Raspberry-től érkező választ. Ezután kiüríti a `Serial` buffert, mielőtt figyelne arra, mivel ezen előfordulhatnak hibás adatok.

Ezek után az Arduino `Serial`-on keresztül elküldi az adatokat a Raspberry-nek, ami `http` lekérést indít a webservice felé. A válasz alapján 0 vagy 1-et küld az Arduino-nak ismét `Serial`-on.

```

1  String wait_for_response(){
2      RFID.end();
3
4      while(Serial.available()) {
5          Serial.read(); // Le kell üríteni a Serial buffer-ét
6      }
7
8      while(!Serial.available()){
9          delay(50);
10     }
11
12     String input = Serial.readString();
```

```

13     input.trim();
14
15     RFID.begin(9600);
16
17     return input;
}

```

3.2. kód. wait_for_response() függvény

A kapott válasz alapján mozgatja a zárat, illetve a led-ek segítségével tájékoztatja a user-t, ugyanazon az elven, mint a teljesen elosztott megoldás esetén. (lásd 2.4)

3.1.5. MQTT

Az IoT-s projektekben gyakran használják az MQTT-t. Először én is ebben gondolkodtam, viszont ahogy jobban utána jártam ennek a technológiának, rájöttem, hogy ehhez a megvalósításhoz nem igazán alkalmas a használata.

Egy MQTT hálózat minden egy brókerből (*MQTT Broker*) és bármennyi kliensből (*MQTT Client*) áll. A bróker témákat (*Topic*) hoz létre (pl. *temperature*), ezeket képzeliük el úgy, mint nyitott chat szobákat. Amikor egy eszköz kíváncsi valamilyen téma irányába, akkor feliratkozhat (*Subscribe*) egy, vagy akár több téma irányába is. Olyan, mint amikor belépünk egy ilyen chat szobába. Amikor valahonnan új adat érhető el, akkor azt, a küldő eszköz valamelyik téma irányában (vagy akár több téma irányába is) elküldi (*Publish*). Az üzenet itt, az HTTP-vel ellentétben nem szöveges, hanem bináris, ezért az adatinkat nem muszáj kódolnunk (pl. *base64*-ben). [34]

Például, egy digitális hőmérő, a `temperature` téma irányába elküldi az aktuális hőméréséket. A gázkazán, aki követi a `temperature`-t, értesítést kap, ez alapján tudja, hogy kell-e fűtenie. Illetve még egy intelligens szellőzés is figyelheti ezt a téma irányában, így előreírhati, hogy kell-e szellőztetni a helyiséget.

A csillagpontos megoldás esetében csak kérdés-válasz kommunikáció zajlik le. Egy Arduino megkérdezi a Raspberry-t, hogy beengedheti-e az adott user-t, majd a Raspberry válaszol neki. Ezért ezt nem gondolom tipikus MQTT felhasználási esetnek.

A továbbfejlesztési lehetőségek fejezetben leírtam egy megoldást, amely esetében már érdemes lehet az MQTT használata. (lásd 5)

3.2. Rest elérés

A Raspberry PI kommunikál a webservice-al Rest API-on keresztül. Egy PUT HTTP metódussal elküldi a zár azonosítóját, illetve JSON formátummá alakítás után a beolvasott adatokat. A visszakapott adatok alapján, amely lehet 1 vagy 0, felkapcsolja a megfelelő ledet, amely jelzi a dolgozónak, hogy hamarosan nyílik a zár vagy sem. 1 válasz esetén felkapcsol egy zöld led, jelezve, hogy hamarosan nyílik az ajtó, majd

Arduino kinyitja a zárat, vár néhány másodpercet, majd visszazárja azt. 0 esetén egy piros ledet kapcsol fel az Arduino, a zárat pedig zárva tartja. Ez a folyamat hasonló az elosztott megoldásban használthoz.

3.3. Tesztelés

Az Arduino esetén ugyanazok a teszteléssel kapcsolatos problémák állnak fenn, mint az elosztott rendszer esetén (lásd 2.5 fejezet).

A Node-RED esetén két féle tesztelésről beszélünk: Először is, tesztelhetjük (és tesztelnünk is kell) az általunk írt Node-okat. Ehhez a `node-red-node-test-helper` nevű csomagra lesz szükségünk, ami tartalmazza a *Mocha test framework*-öt, a *should.js* könyvtárat, illetve minden szükséges segéd függvényt.

3.3.1. Mocha

A Mocha egy sokoldalú JavaScript teszt keretrendszer, aminek az a célja, hogy a lehető legegyszerűbbé tegye a tesztek írását. A tesztek elhelyezésére nincs megkötés, de a fájl nevének `_spec.js`-re kell végződni. A teszt környezet automatikusan megtalálja ezeket, és lefuttatja a benne definiált *test case*-eket. [35]

```
describe("LEÍRÁS", function(){ })
```

Ez a függvény alapvetően nem csinál semmi mást, csupán struktúrát ad a tesztjeinknek. Ez jelenti a *Test Suit*-unkat, amin belül definiáljuk a tesztjeinket. Első paraméter a *Test Suit* neve, ami formázva fog megjelenni a terminálunkban (ez alá lesznek csoportosítva a *Test Case*-eink), második pedig egy függvény, amin belül a tesztjeinket definiálju.

```
it("LEÍRÁS", function(){ })
```

Ez egy teszteset, amit le kell futtatni. Első paraméter egy szöveges leírása a funkciónak, ami csak a programozónak szól, a második pedig a teszt funkció, ami meg lesz hívva.

```
done()
```

Ennek a függvénynek a meghívásával jelezhetjük, hogy végzett a teszt. Ha paraméter nélkül hívjuk meg, akkor az egy sikeres tesztet jelent, ellenkező esetben a teszt sikertelen, és a paraméter kiírásra kerül a konzolra, hogy a programozó lássa, a hiba okát. Továbbá, ha a teszt során bárhol *exception* keletkezik, akkor szintén megbukik a teszt és a hiba üzenet kiírásra kerül.

Továbbá elérhetőek még a következő *hook*-ok:

```
before()
```

Egyszer, bármelyik test case (`it()` függvény) előtt fut le.

```
after()
```

Egyszer fut le, minden test case után.

```
beforeEach()
```

Minden egyes test case előtt lefut.

```
afterEach()
```

Minden egyes test case után lefut.

3.3.2. should.js

A *Mocha* alapvetően nem tartalmaz semmilyen eszközt *assertion*-ök írására, de több csomagot is javasol, amik használhatók. A NodeRED teszt csomagja a *should.js*-t tartalmazza. Ez a könyvtár az olvashatóságot tartja szem előtt. Úgy működik, hogy a JavaScript alap *Object* prototípusát kibővíti egy *should* getter-el, amit meghívva egy *Assertion Object*-et kapunk. Erre az *Object*-re pedig meghívogathatjuk feltételeinket. Például: `user.name.should.be.instanceof(String).and.have.lengthOf(4)`. Ha egy *assertion* megbukik akkor a kód *exception*-t dob. A repository 2020.november 4-e óta archiválva van, tehát ez a csomag frissítéseket már nem kap, de a NodeRED fejlesztői továbbra is támogatják a tesztelő csomagba csomagolt verziót. [36]

3.3.3. Flow tesztelés

A Flow-k tesztelésére nincs hivatalos ajánlás. A manuális tesztelést segítik a debug node-ok, amiket bárhová elhelyezhetünk a flow-ban, illetve az inject node-ok, amik manuálisan el tudják indítani (trigger-eln) az adott flow-t anélkül, hogy meg kéne várunk az eseményt, ami kiváltaná azt. A NodeRED Community két féle megoldást talált a tesztelésre: Az első, hogy létrehoznak egy második flow-t, ami az elsőt teszteli. Ennek az a problémája, hogy a tesztelendő flow valóban fut a Raspberry-n, ahol egyébként is sokkal korlátozottabbak az erőforrások, illetve a potenciálisan hibás output-ok, http kérések stb. valóban megtörténnek. Hasonló a probléma, mint az ArduinoUnit library esetén.

A második megoldás már sokkal jobb. A NodeRED mappájában telepítsük a *mocha* és *should.js* csomagokat. Hozzuk létre a test mappánkat, ebbe helyezzük a test fájlokat. A fájlnév itt is mindegy, de *_spec.js*-re kell végződni. Fontos, hogy itt minden *Test Suit*-ban be kell importálnunk az összes általunk használt node-ot, még akkor is, ha az az alap node-ok közül való. [39]

```
1 const nodes = [
2   require("../node_modules/@node-red/nodes/core/common/20-inject.js"),
3   require("../node_modules/@node-red/nodes/core/function/10-function.js"),
4   // ...
5 ]
```

3.3. kód. Node-ok importálása

Ezután az egyes teszteseteknél minden be kell töltenünk a tesztelendő flow-t. Innen elő hozzáférünk minden node belső állapotához, tehát azokat könnyedén tudjuk tesztelni, illetve a megfelelő eseményeket manuálisan is el tudjuk indítani. Ennek a megoldásnak az egyetlen nagy problémája az, hogy ha a flow-nk más flow-kkal kommu-

nikál akkor azt tudjuk ugyan *mock*-okkal helyettesíteni, de csak nagyon komplikáltan. Továbbá a teszt alatt nincs lehetőségünk a flow-k párhuzamos futtatására sem. [39]

3.4. Lehetséges problémák

Ebben a megoldásban a vezetékes kommunikáció miatt könnyebb megtalálni, hogy melyik zárnál történt meghibásodás. Arduino vagy vezeték hiba esetén a Node-RED-ben láthatjuk, ha az egyik porton nincsen kapcsolat. Ezért, ha más eszköznél van a hiba, akkor azt könnyebb megtalálni. Viszont jelenleg más ellenőrzés az eszközök elérhetőségére nincsen.

Probléma lehet még a túl nagy vezetékes hálózat kialakítása, amely költséges és munka igényesebb lehet. Erre a problémára a továbbfejlesztési lehetőségek fejezetben igyekeztem megoldást találni. (lásd 5)

4. fejezet

A két megoldás összehasonlítása

Mindkét megoldás ugyanazzal a webservice-al, Rest API-on keresztül kommunikál, HTTP metódusok segítségével, JSON formátumban, emellett ugyanazokat a végpontokat használják. A mikrokontrollerek programkódja pedig C++-ban íródott.

Vezetékelés

A csillagpontos megoldásban az Arduino-k vezetékes kapcsolaton keresztül kommunikálnak a Raspberry-vel, míg a teljesen elosztott megoldásban az ESP-k csak az olvasóval, illetve a motorvezérlővel vannak összekötve.

REST elérés

A csillagpontos megoldásban csak a Raspberry kommunikál közvetlenül, a Rest API-on keresztül a webservice-al.

Míg az elosztott megoldásban minden egyes ESP eléri azt.

Költségek

A csillagpontos megoldás összességében drágább a Raspberry és az Arduino miatt. Viszont a Raspberry lecserélhető valamilyen, hasonló tulajdonságokkal bíró SBC-re (Single Board Computer), ezzel is csökkenthető a beléptető rendszer költsége. Ha az eszközök messze vannak egymástól, a vezetékelés is költséges lehet. Természetesen az Arduino is kicserélhető bármilyen olcsóbb eszközre, amelynek van elég pin-je.

Az elosztott megoldásnál a stabil WiFi elérés biztosítása jelenthet többlet költséget.

Összességében az, hogy melyik megoldás költséghatékonyabb leginkább a hely méretétől, adottságaitól, illetve a router hatótávolságától függ.

Megbízhatóság

A csillagpontos módszernél több, de egyszerűbb eszköz van, amik kevésbé hajlamosak a meghibásodásra, emiatt ez a megoldás megbízhatóbbnak bizonyulhat. Az elosztott megoldás esetén a 2,4 GHz-es WiFi érzékenyebb lehet a zavaró jelekre pl. egyéb IoT eszközökből érkező jelek. Továbbá az ESP-8266 hajlamos elveszíteni a WiFi jelet, illetve egyéb csatlakozási problémák is előfordulhatnak.

Karbantarthatóság

A backend jelenleg nem ellenőrzi, hogy egy eszköz meghibásodott-e. A csillagpontos megoldásnál a Raspberry ezt tudhatja ellenőrizni és értesíteni az illetékeseket.

A vezetékelést nem tekintem karbantarthatósági problémának, mivel a javításuk egyszerű.

Kártya olvasás

Mindkét megoldásban azonos típusú RFID olvasót és tag-eket használtam. Nem találtam az általam használt (RDM6300) olvasónál olcsóbb eszközt.

Motor vezérlés

A zárak mozgatásának szemléltetésére egy egyenáramú motort használtam, amelyet egy L293 D (2.1.2) motorvezérlővel irányítottam mind két esetben. Természetesen, más fajta zárakkal is megvalósíthatóak ezek a megoldások, viszont ehhez kódbeli változtatásokra lesz szükség.

Visszajelzés

Mindkét megoldásban egy zöld led jelzi, hogy a felhasználó beléphet az adott ajtón, illetve egy piros led, ha nem léphet be rajta. Abban az esetben is a piros led világít, ha hiba történt az azonosítás során.

Programozás

Mindkét megoldásnál a programkód C++-ban íródott, ehhez az Arduino IDE-t használtam. Kivéve a Raspberry Pi esetén, ahol NodeRED-ben valósítottam meg a működést. Az utóbbi miatt előfordulhat, hogy JavaScript-et is kell használnunk.

5. fejezet

Továbbfejlesztési lehetőségek

Ebben a fejezetben azoknak a problémáknak a megoldására próbáltam megoldást keresni, amelyek felmerültek a megvalósítási folyamat alatt.

Meghibásodások figyelése

Az elosztott megoldásban jelenleg, ha egy ESP meghibásodik, azt nehezen lehet észrevenni, illetve megtalálni, hiszen nincs benne olyan eszköz, ami erre figyelne. Ezért, ha az ESP-k pl. egy Raspberry-vel is kommunikálnának, akkor a Raspberry már tudná figyelni az ESP-k működését, így meghibásodás esetén tudna szólni a megfelelő illetékesnek, és még azt is meg tudhatja mondani, hogy melyik eszköz hibásodott meg, hiszen az ESP-k saját azonosítóval rendelkeznek, ezzel növelte a megoldás javíthatóságát.

Vezetéknélküli csillagpontos megoldás

A két általam megvalósított megoldás között „félúton” helyezkedik el ez a megoldás. Továbbra is egy SBC (pl. Raspberry) az egyetlen eszköz REST eléréssel, de a vezetékes kapcsolat helyett WiFi-n kapcsolódnának ESP-01-ek hozzá. A Raspberry az előző továbbfejlesztési lehetőség alapján tudna figyelni az esetleges meghibásodásokra.

JWT

A kód jelenleg nem küld JWT azonosítót a webservice-nak, de később erre is szükség lesz. Ez úgy tud működni, hogy az eszköz bejelentkezik a `webservice/login` végpontján, és tárolja az innen kapott TOKEN-t. Ezután minden üzenet fejlécében el kell azt küldeni.

MQTT-s megvalósítás

Jelenleg az egyik megvalósítás sem használ MQTT-t, viszont, ha a zár nyitására figyelne több eszköz is, akkor már több értelme lenne a használatának. pl. felkapcsolhatna a lámpa az adott helyiségben, vagy akár kamerák fordulhatnának az ajtó felé, hogy követni lehessen, ki és mikor járt bent. stb.

Saját eszközök

Akár saját eszközöket is lehetne készíteni, amelyek belül, egy ESP-01-et és az olvasót tartalmazzák, kívülről, pedig csak a szükséges csatlakozók kivezetéseit. Így egy szinte bárki számára egyszerűen összeszerelhető és beüzemelhető rendszert lehetne alkotni. Mind a két általam bemutatott megvalósítással működhetnek ezek az eszközök.

5.1. Egyéb megvalósítási lehetőségek

Ebben a fejezetben további esetleges megvalósítási lehetőségeket kerestem a beléptető rendszerre. Ezek leginkább csak ötletek.

Bluetooth eszközök

Vannak olyan ESP-k, illetve RFID olvasók, amelyek képesek Bluetooth kapcsolatra, ezzel a megoldással esetleg még több vezetékes összeköttetést ki lehetne váltani. Viszont ezzel valamelyest nőne a megvalósítás költsége, illetve esetleg a Bluetooth kapcsolat miatt a megbízhatósága is csökkenhet.

ESP-k kihagyása

Megvalósítható a csillagpontos megoldásomnak egy olyan verziója is, ahol nincsenek ESP-k vagy Arduino-k, és az olvasók, valamint a zárak közvetlenül kapcsolódnak a Raspberry-hez, viszont úgy gondolom, hogy gyakorlatban ennek a megoldásnak csak akkor lenne értelme, ha a leolvasók (és esetleg a zárak is) Bluetooth-on tudnak kapcsolódni, és nem kell annyi vezetéket kihúzni. Ebben az esetben viszont, a Bluetooth-os leolvasók lehet, hogy annyi többlet költséget jelentenének, hogy nem spórolnánk semmit.

Elektromágneses ajtózár

Mindkét megoldásomban egy motor nyitotta, zárta a zárat, ez volt számomra a leg könnyebben elérhető megoldás, mert már eleve rendelkezésemre állt a motor és a motorvezérlő is. Viszont valószínűleg egy valós használatnál egy elektromágneses zár olcsóbb,

gyorsabb és egyszerűbb megoldás lehetne, ezért ennek a programkódját is elkészítettem szemléltetésképpen.

```
1 #define LOCK 5
2
3 setup() {
4     //...
5     pinMode(LOCK, OUTPUT);
6     digitalWrite(LOCK, HIGH);
7 }
8
9
10 if(status == "1") {
11     digitalWrite(GREEN_LED, HIGH);
12     digitalWrite(LOCK, LOW);
13     delay(OPEN_TIME);
14     digitalWrite(LOCK, HIGH);
15     digitalWrite(GREEN_LED, LOW);
16 }
17
18 }
```

5.1. kód. Elektromágneses zár vezérlő

A kódban természetesen nincs szükségünk a motor pin-jeinek definíciójára, illetve a `motor()` függvényre sem. A mágneses zár általában 12, 24, vagy 48 Volton működik, ezért egy relén keresztül tudunk áramot adni neki, és a mikrokontroller egyik pin-jével tudjuk vezérelni azt (kódban 5-ös pin). Ezt a pin-t a `setup()` függvényben `OUTPUT`-ra kell állítanunk, és `HIGH` jelet kell kapcsolnunk.

A `loop()` függvényben, a beengedés kódja lényegesen egyszerűsödik, hiszen csak `LOW` jelet kell kapcsolni a relét vezérlő pin-re, annyi ideig, amíg szeretnénk a zárat nyitva tartani. A `HIGH` jel visszakapcsolásával ismét bezárjuk a zárat.

Szolenoid zár

Az elektromágneses zár alternatívjaként használható. Úgy működik, hogy a zárnyelvre egy rugó van erősítve, és egy elektromágnes, ha áramot kap ez a rugó ellen dolgozik. Szokás még úgy is megvalósítani, hogy a zár nyelv függőlegesen van beépítve, így az elektromágnes a gravitáció ellen dolgozik. Két típusa van: Záró és nyitó. Az elnevezés azt jelenti, hogy mi történik, amikor az elektromágnes áramot kap. [37]

Bekötést tekintve ezek a zákok is nagyobb feszültséget használnak, mint a mikrokontrollerek, így itt is relén keresztül kell áramot adni nekik. Bekötés mind két típusnál megegyezik az elektromágneses zárral.

Az első esetén a rugó visszahúzva tartja a nyelvet, de az elektromágnes „kitolja” azt. Ez a megoldás azért jó, mert ha elmegy az áram, például áramkimaradás vagy tűz

miatt, akkor az ajtó biztosan kinyílik. Gyakran használják vészki járatok esetén. [38] Ez pontosan azzal a programkóddal működne, mint az elektromágneses zár.

A nyitó szolenoid zár fordítva működik. A rugó vagy gravitáció zárva tartja az ajtót mindaddig, amíg áramot nem kap. Ez azért jó, mert az előző változattal (és az elektromágneses zárral) ellentétben nem kell folyamatos feszültséget biztosítanunk neki. Olyan helyeken szokás használni, ahol az elsődleges cél a biztonság. [37] Ennél a zárnál a programkódot úgy kellene változtatnunk, hogy pontosan megfordítjuk, hogy mikor küldünk *HIGH* és *LOW* feszültséget a relé vezérlő lábára.

Kódzás azonosítás

A kódzás azonosítás Arduino IDE-ben a *Keypad library*-nek köszönhetően nem túl nehéz feladat. Ez a könyvtár a mátrix stílusú billentyűzetek-hez van kitalálva. A billentyűzet bekötéséhez nincs szükség ellenállásokra, diódákra, mert a könyvtár automatikusan **INPUT_PULLUP**-ra állítja az érintett pin-ek pinMode-ját.

Ahhoz, hogy használni tudjuk a Keypad könyvtárat, először a *Library Manager*-ben kell megkeresni, majd installálni azt.

Először is, meg kell adnunk, hogy hány darab sora, illetve oszlopa van a billentyűzetünknek, hiszen ebből is létezik több fajta. Majd ennek megfelelően, el kell készítenünk annak mátrixos megfelelőjét. Ezek után le kell írnunk, hogy sorrendben mely pinek tartoznak a sorokhoz, illetve az oszlopokhoz. Legvégül pedig ezeknek az adatoknak a segítségével meg kell hívnunk a Keypad konstruktort.

```
1 const byte rows = 4;
2 const byte cols = 3;
3 char keys[rows][cols] = {
4     {'1','2','3'},
5     {'4','5','6'},
6     {'7','8','9'},
7     {'#','0','*'}
```

```
8 };
9 byte rowPins[rows] = {5, 4, 3, 2};
10 byte colPins[cols] = {8, 7, 6};
11 Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, rows,
cols );
```

Ennek a könyvtárnak nagyon sok, hasznos függvénye van, mint a char visszatérési értékű **getKey()**, amely visszaadja a lenyomott gombot, ha van és közben nem blokkol más függvényeket.

```
1 void setup(){
2     Serial.begin(9600);
3 }
```

```

5   String text = "";
6   void loop(){
7     char key = keypad.getKey();
8     if (key != NO_KEY){
9       switch(key) {
10         case '*':
11           text = "";
12           break;
13         case '#':
14           // Itt el kell küldeni a text-et
15           text = "";
16           break;
17         default:
18           text += key;
19           break;
20       }
21     }
22
23     // timeout kezelés
24 }
```

A beírt kódot továbbra is a `worker_rfid` mezőben kell elküldeni. Érdemes egy `timeout` kezelést hozzáadni, ami azt jelenti, hogy ha a user nem nyom meg semmilyen gombot egy adott időn belül, akkor a `text` üresre lesz állítva.

Egyéb azonosítási lehetőségek

Mindkét megvalósításban egy RFID-s azonosítást valósítottam meg, a backend vár egy `worker_rfid` property-t a *request body*-ban, de ez lehet bármilyen adat, amely egyedileg azonosítja a dolgozót, pl. bináris adat (base64 kódolva) egy újlenyomat olvasótól vagy retina szkennertől, vagy akár küldhetünk neki JSON object-et is, hogy felhasználónév-jelszóval azonosítsuk a dolgozónkat.

A backend limitáltsága, hogy egy dolgozóhoz csak egy azonosító tartozhat, ezért nem tudunk egy dolgozóhoz felvenni RFID-t és újlenyomatot is felvenni. Ez a továbbfejlesztés a webservice oldaláról tervben van.

Összegzés

Összességében, mivel még ezelőtt nem dolgoztam ESP-kkel, Raspberry-vel, RFID olvasóval, illetve NodeRED-el sem, ez egy elég érdekes és tanulságos projekt volt, amiből rengeteg új tapasztalatot sikerült szerezniem.

Megtudtam, hogy nagyon sokféle ESP létezik, amely elég nagy szabadságot kínál a funkciók terén. Viszont Magyarországon sajnos elég kevés érhető el közülük, amely pedig nehezítheti az egyes feladatok ESP-vel történő megoldását. Az RFID kártyák kiválasztásánál oda kell figyelni az olvasó olvasási frekvenciájára. A Raspberry PI 4 egy elég sokrétűen használható és programozható eszköz, amivel az előítéleteim ellenére egészen könnyen boldogultam.

Az eddigi Arduino IDE-s tudásomat is sikerült tovább fejlesztenem, az RFID olvasó és az ESP miatt. Sokat tanultam a SoftwareSerial-ról, arról, hogyan kell wifi hálózatra csatlakozni, illetve, hogyan kell HTTP kéréseket küldeni, majd az azokra érkező választ fogadni.

Úgy gondolom, hogy a dolgozatomban szereplő megoldásokat a minden nap használatra is meg lehetne valósítani. Valószínűleg nem pontosan azokkal az eszközökkel, mint amiket én használtam, ezért igyekeztem a legtöbb eszköz helyettesítésére példákat keresni.

Az Arduino library-nek köszönhetően a kód rendkívül hordozható, illetve más ESP-kkel is használható, minimális módosításokkal (pl. Wifi kezelés). Valamint más zártípusok esetén a programkódot biztosan módosítani kell, hogy azt a zárat vezérelni tudjuk.

Jelenleg úgy gondolom, hogy a szakdolgozatom megvalósítása közben felmerülő kérdések és problémák sem vették el a kedvemet a hardverekkel történő munkától. Illetve, hogy a jövőre való tekintettel értékes tapasztalatokat szereztem, remélem, hogy az itt megszerzett tudásomat és tapasztalataimat tovább tudom bővíteni.

Köszönet nyilvánítás

Mindenképpen szeretném megköszönni az egyetemi oktatóimnak a négy év alatt tőlük kapott rengeteg tudást, tapasztalatot és segítséget.

Külön köszönetet szeretnék mondani Dr. Tajti Tibor tanár úrnak a segítségért, amelyet a szakdolgozatom elkészítése, illetve az egyetemen töltött éveim során kaptam tőle.

Dombi Tibor Dávidnak, hogy használhattam az általa írt webservice-t, illetve, hogy bármikor fordulhattam hozzá segítségért.

Köszönöm a hallgatótársaimnak, szaktársaimnak, barátaimnak, akik között mindig voltak olyanok, akik segíteni tudtak a szükséges pillanatokban.

Végül szeretném megköszönni a szüleimnek, családomnak, hogy bíztattak, mellettettem álltak és végig támogattak az egyetemi éveim alatt.

Irodalomjegyzék

- [1] Repository
<https://github.com/jalexandra/yii8yw-thesis>
- [2] DOMBI TIBOR DÁVID: *Webservice menedzselhető beléptetőrendszerhez*, Eszterházy Károly Katolikus Egyetem, Eger, 2022.
<https://github.com/dombidav/hl5u4v-thesis>
- [3] Webservice telepítő varázsló
<https://github.com/dombidav/hl5u4v-thesis/blob/main/README.md#with-wizard>
- [4] ESP Specifikus függvények
<https://arduino-esp8266.readthedocs.io/en/latest/libraries.html#esp-specific-apis>
- [5] JWT Handbook
Sebastián E. Peyrott, Auth0 Inc (Version 0.14.1, 2016-2018)
- [6] A RFID tag-ek felépítése
<https://rfid4u.com/dig-deep-construction-of-rfid-tags/>
- [7] RFID típusok
<https://www.makeuseof.com/tag/technology-explained-how-do-rfid-tags-work/>
- [8] Tag típusok
<https://hu.wikipedia.org/wiki/RFID>
- [9] RDM6300 Specifikáció
<https://www.handsontec.com/dataspecs/module/RDM6300.pdf>
- [10] L293D Specifikáció
<https://www.ti.com/lit/ds/symlink/l293.pdf>
- [11] Arduino Motor Shield
<https://store.arduino.cc/products/arduino-motor-shield-rev3>

[12] Espressif ESP8266

<https://www.espressif.com/en/products/socs/esp8266>

[13] What is CH340

<https://www.seeedstudio.com/blog/2020/09/30/usb-serial-what-is-ch340/>

[14] Arduino IDE setup

<https://help.ubidots.com/en/articles/928408-program-the-esp8266-with-the-arduino-ide-in-3-simple-steps>

[15] SPI Flash Modes

<https://docs.espressif.com/projects/esptool/en/latest/esp8266/advanced-topics/spi-flash-modes.html>

[16] Software Serial Library

<https://docs.arduino.cc/learn/built-in-libraries/software-serial>

[17] ESP8266WiFi Library

<https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html>

[18] Arduino Unit Testing

https://stackoverflow.com/questions/780819/how-can-i-unit-test-arduino-code#comment10949156_791519

[19] PlatformIO Unit Test

<https://docs.platformio.org/en/stable/advanced/unit-testing/index.html>

[20] Raspberry PI OS

<https://www.raspberrypi.com/software/>

[21] Node-RED Raspberry-re (OpenJS Foundation & Contributors)

<https://nodered.org/docs/getting-started/raspberrypi>

[22] Serial portok Node-RED-ben (OpenJS Foundation & Contributors)

<https://flows.nodered.org/node/node-red-node-serialport>

[23] RFID

<https://ident.hu/rfid>

[24] ESP8266

<http://faragocsaba.hu/esp8266>

- [25] Alternatívák Raspberry PI helyett
<https://pcworld.hu/tippek/4-alternativa-a-raspberry-pi-helyett-161002.html>
- [26] SoftwareSerial Library
<https://docs.arduino.cc/learn/built-in-libraries/software-serial>
- [27] RFC 4252 - The Secure Shell (SSH) Authentication Protocol
T. Ylonen, C. Lonvick, The Internet Society (2006)
- [28] What is an SSH Key?
<https://www.ssh.com/academy/ssh-keys>
- [29] How To Set Up Authorized Keys
https://wiki.qnap.com/wiki/How_To_Set_Up_Authorized_Keys
- [30] Arduino Uno
<https://store.arduino.cc/products/arduino-uno-rev3>
- [31] Node-RED Flow (OpenJS Foundation & Contributors)
<https://nodered.org/about/#flow-based-programming>
- [32] Creating Nodes (OpenJS Foundation & Contributors)
<https://nodered.org/docs/creating-nodes/first-node>
- [33] What is IoT
<https://www.oracle.com/internet-of-things/what-is-iot/>
- [34] MQTT
<https://mqtt.org/>
- [35] Mocha
<https://mochajs.org/>
- [36] Should.js
<https://github.com/shouldjs/should.js>
- [37] Szolenoid zár
<https://doorcontroldirect.co.uk/31-solenoid-door-locks-motor-locks>
- [38] Elektromos zár típusok
<https://www.takigen.com/products/list/L-002>
- [39] Flow Tesztelés
<https://www.technicalfeeder.com/2021/02/how-to-write-node-red-flow-test/>

NYILATKOZAT

Alulírott Jánbor Alexandra büntetőjogi felelősségem tudatában kijelentem, hogy az általam benyújtott, Menedzselhető beléptető rendszer fizikai megvalósítása című szakdolgozat (diplomamunka) önálló szellemi termékem. Amennyiben mások munkáját felhasználtam, azokra megfelelően hivatkozom, beleértve a nyomtatott és az internetes forrásokat is.

Tudomásul veszem, hogy a szakdolgozat elektronikus példánya a védés után az Eszterházy Károly Katolikus Egyetem könyvtárába kerül elhelyezésre, ahol a könyvtár olvasói hozzájuthatnak.

Kelt: Eger , 2023. év április hó 02. nap.

..... Jánbor Alexandra

aláírás