SZAKDOLGOZAT FELADAT

Ágó Bernát (VKMZ8A)

mérnök informatikus hallgató részére

Moduláris időjárásállomás tervezése és fejlesztése

Az egyre gyorsuló világunkban egyre több esetben fontos számunkra az aktuális időjárás, legyen az akár a hőmérséklet, vagy a páratartalom értéke, kinti, de akár a benti, a lakásban. Az utóbbi években az egyre erősödő klímaváltozás miatt ennek az igénynek a kielégítése kiemelkedő jelentőséggel bír, hiszen ezen értékek gyorsan változnak. Ennek a projektnek a célja egy olyan egyszerű időjárásállomás tervezése fejlesztése, ami könnyen használható és bárki számára elérhető.

A legfontosabb elvárások:

- Mobil működés akkumulátorról
- Modularitás, könnyen adaptálható legyen másik szenzorokra
- Olcsó alkatrészek használta a hozzáférhetőség miatt
- Biztonság és adatvédelem
- Adatok tárolása központi szerveren
- Több eszköz együttes működése
- Adatok, statisztikák lekérdezése okostelefonról, WiFin keresztül

A hallgató feladatának a következőkre kell kiterjednie:

- Mutassa be a szenzorok legfontosabb típusait is technológiákat
- Ismertesse a szenzoradatok felhasználásában rejlő lehetőségeket
- Válassza ki a szükséges szenzorokat és technológiákat
- Tervezzen saját rendszert
- Valósítsa meg azt, és tesztelje az elkészült megoldását
- Dokumentálja az elért eredményeket
- Vázolja a tovább fejlesztés lehetőségeit

Tanszéki konzulens: Schulcz Róbert, BME Hálózati Rendszerek és Szolgáltatások Tanszék
Külső konzulens: -

Budapest, 2021.szeptember 17.

Dr. Imre Sándor
egyetemi tanár
tanszékvezető

Konzulensi vélemények:
Tanszéki konzulens: □ Beadható, □ Nem beadható, dátum: aláírás:
Külső konzulens: □ Beadható, □ Nem beadható, dátum: aláírás:

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék
1117. Budapest, Magyar tudósok körútja 2. I. Ép. B. 121.
Telefon: 463-3261 Fax: 463-3263
URL: http://www.hit.bme.hu

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék

Ágó Bernát

MODULÁRIS IDŐJÁRÁSÁLLOMÁS TERVEZÉSE ÉS FEJLESZTÉSE

IoT Eszközök

KONZULENS

Schulcz Róbert

BUDAPEST, 2021

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Ágó Bernát**, 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, 2021. 12. 09.

...…….
Ágó Bernát

Összefoglaló

Ebben a dolgozatban bemutatom egy hobbielektronikai eszközöket alkalmazó időjárásállomás tervezését, fejlesztését és implementálását. A projekt célja egy olyan rendszer létrehozása, ami könnyen kezelhető, önállóan működtethető, valamint az ezen a területen járatosak számára könnyen módosítható.

A projekt három fő részből áll. A rendszer szíve az érzékelő egység, ami az ATmega328P és az ESP8266 mikrokontrollerekre, valamint az ezekhez kapcsolt szenzorokra épül. A második, nem kevésbé fontos rész egy backend szerver, ami megteremti a kapcsolatot a mérőeszköz és a frontend között. A harmadik része a felhasználói interfész, amiről az egész rendszer kezelhető.

A fejlesztés során számtalan tényező megváltozott, a munka léptéke jelentősen megnövekedett. Több komponens teljesen újra lett tervezve, hogy dinamikusan illeszkedjen a projekt többi részéhez, és hogy a rendszer végig fenntartható és bővíthető maradjon.

A téma tartalmazza többek között az alábbi technológiákat és megoldásokat: WiFi, mind kliens, mind szerver oldalról, konténerizált rendszerek Docker segítségével, web backend technológia Go és az arra épülő Fiber használatával, androidos alkalmazás, akkumulátoros működés és még sok más.

Abstract

In this paper I present the design, development and implementation of a weather station using hobbyist electronic components. The aim of the project is to create a system that is easy to use, capable of operating by itself, and easily modifiable by people proficient in this field.

The project consists of three main parts. At the heart of the system is the sensor unit, which is based on the ATmega328P and ESP8266 microcontrollers and the sensors connected to them. The second, no less important part is a backend server, which creates the connection between the measuring device and the frontend. The third part is the user interface from which the whole system can be operated.

A lot has changed during the development and the scope of the work has increased considerably. Several components have been completely redesigned to fit dynamically with the rest of the project and to ensure that the system remains sustainable and extensible throughout.

The subject matter includes technologies and solutions such as WiFi, both client and server side, containerised systems using Docker, web backend technology using Go and Fiber built on top of it, application based on android, battery operation and many more.

1 Bevezetés

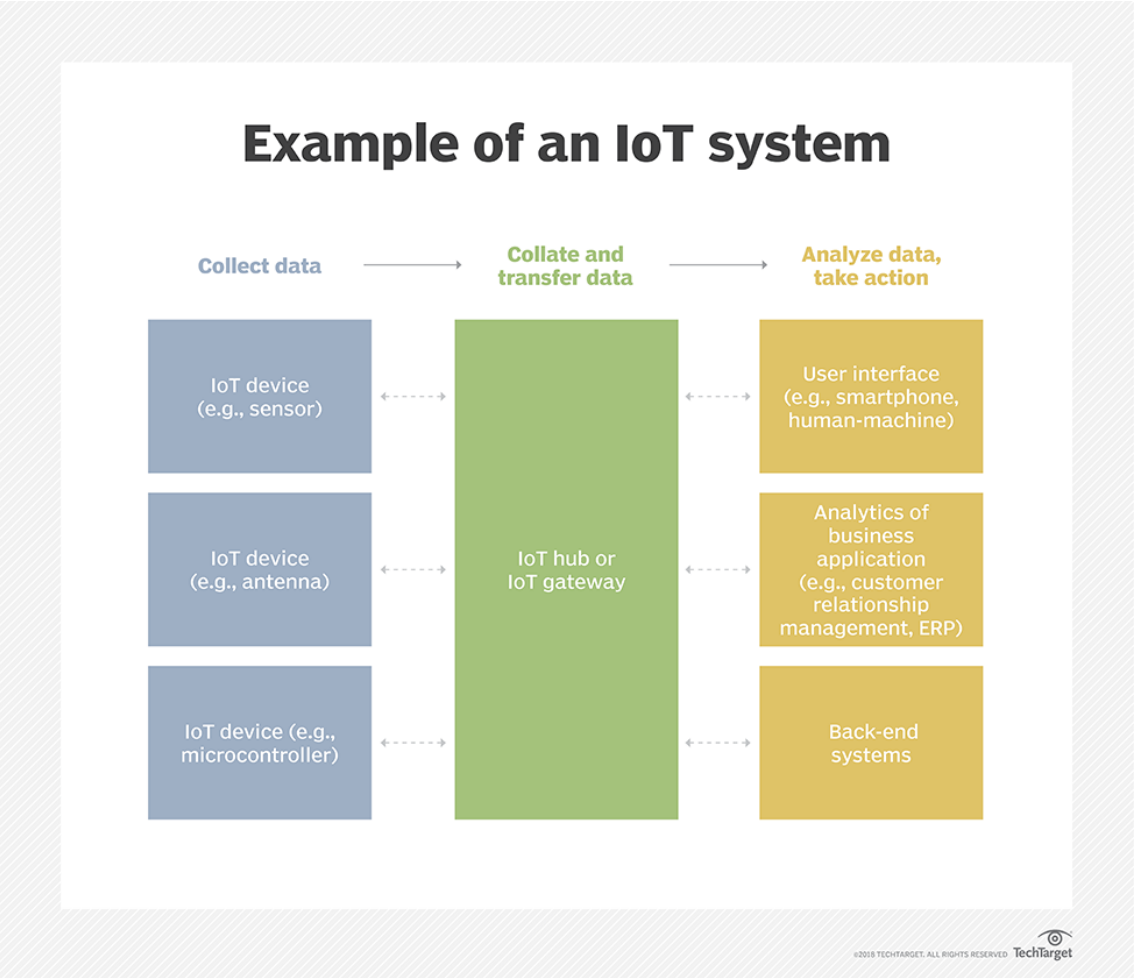
Részben azért döntöttem úgy, hogy ezt a témát szeretném kidolgozni egy szakdolgozat keretein belül, mert már sok éve foglalkozom hasonló dolgokkal a szabadidőmben. Így ezt nagyon jó lehetőségnek láttam, hogy egyetemi keretek között szánjak rá több időt. A munka, talán a mostani trendeknek szembemenően, egy alacsony szintű tervezési feladat, ami azt jelenti, hogy minél kevesebb előre összerakott, szoftveres és hardveres alkatrészt használok, helyette inkább megpróbálom ezeket a lehető legalapvetőbb elemekből összerakni. Ez több okból is fontos. Egyrészt, ez lehetőséget ad nekem arra, hogy részletekbe menően megismerjem a használt technológiákat, ezáltal értékes tudást szerezve. Másrészt, így sokkal testre szabhatóbb lesz a végeredmény, olyan optimalizációkat tudok használni, ami magasabb szinten nem lenne lehetséges.

1.1 Az IoT

Mi is az az IoT? Szó szerinti magyar fordítása a „Dolgok Internete”, de a jelentése ennél jóval mélyebb. Ezt a kifejezést olyan dolgokra használjuk, amik általában kicsik, modulárisak, olcsók, de legfőképpen interneten vagy más vezeték nélküli technológián keresztül egymással kommunikálnak, így együtt egy rendszert alkotnak. Példája ilyen eszközöknek egy távolról vezérelhető termosztát, egy okos kisállateledel adagoló, vagy akár egy időjárás állomás is. Az IoT világában igen gyakran használt az „okos” kifejezés, ez általában az előbb felsorolt tulajdonságokkal jelent egyet.

1.1.1 Hogy néz ki egy IoT rendszer?

Az IoT fogalom három fő folyamatot különít el egy rendszer működésének leírására. Ezek az adatgyűjtés, adatfeldolgozás valamint az adatértelmezés és cselekvés. Az adatgyűjtés fázisban az érzékelők méréseket végeznek, majd az ebből kinyert adatokat minimális változtatásokkal továbbküldik a feldolgozó egységnek. Ezután az adatok egy központi eszközben összegyűjtésre és tárolásra kerülnek. Végül, akár ugyanitt, vagy egy erre specializált komponensben a rendszer eldönti, hogy mit fog cselekedni. Ez lehet az adatok kimutatása egy felhasználónak, az adatok további elemzése statisztikák készítése céljából, vagy cselekvésre képes eszközök mozgósítása. [1]



1.1. ábra: Példa egy IoT rendszerre [2]

Egy ilyen folyamatra példa az okos kisállatetető. Az etető egy kamerával, nyitható és csukható adagolóval, valamint WiFi képes mikrokontrollerrel van felszerelve. Az adatgyűjtés fázis az, hogy a kamera által felvett képen megnézzük, hogy van-e még a tálban étel, az idő mérése, valamint, hogy a gazda az okostelefonján megnyomja-e az „Etetés most” gombot. Ez feldolgozásra, illetve tárolásra kerül egy központi szerveren. Az értelmezés fázisban a központi egység utasítást ad az adagolónak, hogy egy adag ételt porciózzon ki, ha a tálban lévő már elfogyott, és vacsoraidő van.

1.1.2 IoT technológiák

Az IoT rendszerek az utóbbi évtizedekben végbement hatalmas technológiai fejlődésnek köszönhetően jöhettek létre. Ilyen például a vezeték nélküli technológiák, a WiFi, a Bluetooth és a Bluetooth Low Energy, az 5G, valamint a szabadon használható rádiós frekvenciákon alapuló egyedi protokollok [3]. Ezen technológiák közül mindegyiknek

meg vannak a saját előnyei és hátrányai, amiket figyelembe véve olyan eszközöket tervezhetünk, amik kiaknázzák a technológiák által nyújtott lehetőségeket.

A szenzorok is hatalmas fejlődésen mentek keresztül, ma már szinte minden mérhető dologra létezik hozzáférhető áron IoT szenzor, ami egy okos eszközre kapcsolható. Ilyenek például a hőmérő, páratartalommérő, nyomásérzékelő, giroszkóp, légszennyezettségmérő, talajnedvességmérő, fényintenzitásmérő és még sok más.

Az adatelemzésre is számtalan módszerünk van. Az utóbbi időkben leginkább emlegetett ezek közül a Mesterséges Intelligencia, vagy AI. Nem véletlenül, a mesterséges intelligenciára nagyon nagy szükségünk van, ha az úgynevezett Big Data kategóriába tartozó adathalmazokat is fel akarjuk dolgozni. A Big Data fogalom olyan adathalmazok feldolgozásával foglalkozik, amelyek tradicionális módszerekkel, komplexitásuk és méretük miatt, átláthatatlanok. Az IoT eszközök által generált adatok is ilyen halmazt alkotnak, a sok különböző szenzornak köszönhetően. Az AI ezen adatok értelmezésében segít nekünk, a gépi tanulás módszerével összefüggéseket és kapcsolatokat tud felfedezni, és ezáltal további információt tud kinyerni. Erre egy példa a mezőgazdaság, ahol a változók mennyisége miatt szükség van ezekre a módszerekre.[4]

1.1.3 Az IoT kritikái

Az IoT világában sincs minden kompromisszumok nélkül, az IoT-t rengeteg kritika éri, ezek sokszor az IoT elveivel kapcsolatos problémákat fogalmaznak meg.

Az első ezek közül az adatgyűjtéssel kapcsolatos. A fő probléma itt az, hogy rengeteg adat, ami eddig nem létezett digitálisan, az most mind össze van gyűjtve, ráadásul egy helyen tárolva. Ha pedig ezek az adatok léteznek, az azt jelenti, hogy el is lehet őket lopni. Ez pedig adott esetben lehetővé teszi olyan információk kinyerését, amikre nem is gondolnánk. Ha ezek illetéktelenek kezébe jutnak, sokkal nagyobb gondban lehetünk, mintha nem alkalmaztunk volna IoT technológiákat.

Szintén nagyon fontos téma a biztonság. Mivel az IoT eszközök természetükből fakadóan kicsik, olcsók és sok van belőlük, nagy biztonsági kockázatot jelentenek. Ha a támadók akár egy eszközt fel tudnak törni, akkor potenciálisan az egész otthoni hálózat, vagy IoT rendszer kompromittálódhat. Minél több eszközünk van és minél kevesebb fejlesztési erőforrást szánunk a biztonság fejlesztésére, ennek az esélye annál nagyobb.

Ezeken felül kiemelten fontosnak tartom még az autonómiával kapcsolatos aggodalmakat. Itt az a probléma, hogy nem tudjuk, hogy az IoT eszközök által gyűjtött adatokkal pontosan mi történik. Gyakran a feldolgozás egy külső szerveren zajlik, amire nem látunk rá. Ez a biztonsági szempontból sebezhetőbbé teszi a rendszert, valamint rákényszerít minket, hogy rábízzuk az adatainkat egy harmadik félre, aki adott esetben ezzel visszaélhet.

Ezen szempontok miatt döntöttem úgy, hogy olyan rendszert tervezek, amiben külön figyelmet fordítok a fent említettekre, mert számomra ezek kiemelkedő fontossággal bírnak.

1.2 Megismerkedés az eszközökkel

A kezdő koncepció: WiFis érzékelőállomás megvalósítása a következő eszközökkel:

Hardver:

- Arduino Uno mikrokontroller
- DHT22 hőmérséklet és páratartalom mérő
- ESP8266 WiFi képes mikrokontroller modul
- generic FTDI breakout board

. egyéb kisebb passzív és aktív komponensek

Szoftver:

- Arduino IDE
- ESP generic download tool

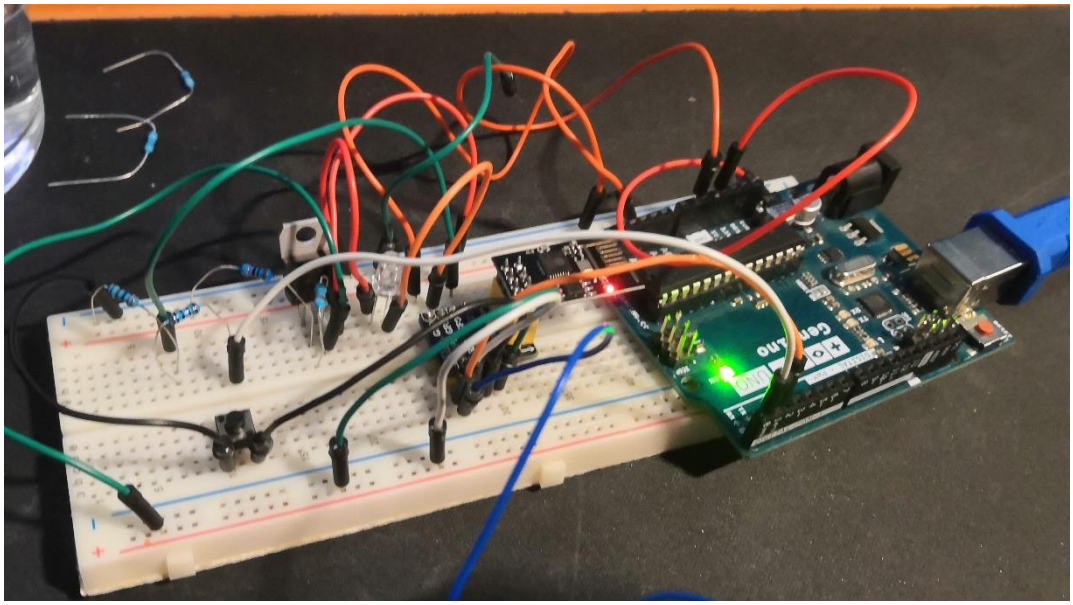
Az ESP8266 mindössze 8 lábbal rendelkezik, amiből csak 4-et használhatunk bármiféle adat átvitelére, ezért van szükségünk az Arduino-ra is, mivel annak sok port-ja van, ehhez tudjuk majd a szenzorokat és egyéb eszközöket kapcsolni.

Az ESP és az Arduino között UART protokollt használva soros kommunikáció valósítható meg, így a számításokat és nagyobb műveleteket az Arduino-n lehet végezni, az ESP csak az adatok vezeték nélküli küldésére lesz használva.

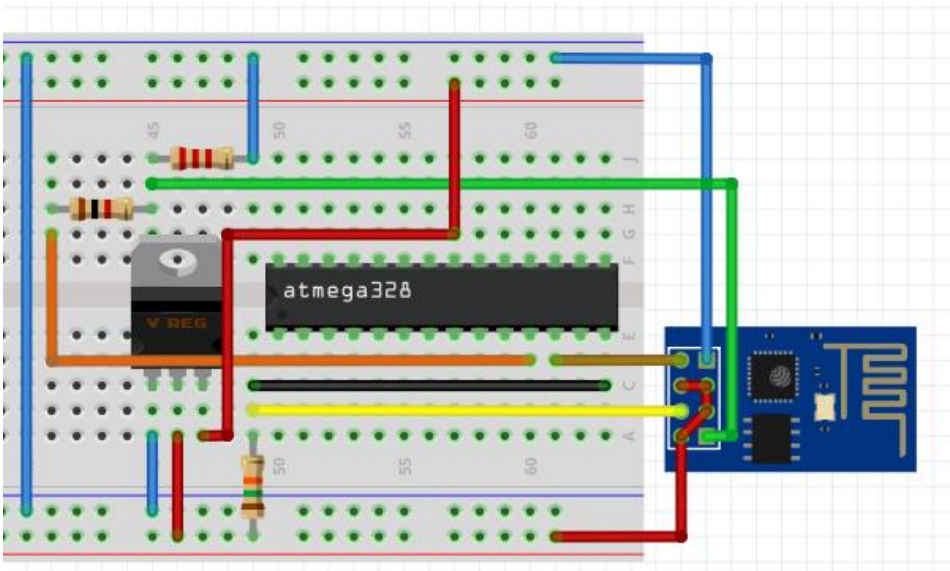
1.3 Első lépések

Mivel még nem érkezett meg minden eszköz, jelenleg csak az ESP és az Arduino áll rendelkezésre. Mivel az Arduino-nak sok lába van, ezért meg lehet azt tenni, hogy ezen keresztül kommunikáljon az ESP a számítógéppel, mivel az ESP-n nincsen USB port. Ez

egy kicsit nehézkes, mivel a szoftveres és hardveres soros kommunikáció közt problémák léptek fel, emiatt a számítógépre néha sérült adat érkezett. Úgy tűnt, hogy ez kizárólag az Arduino és a gép között lépett fel, így a küldött parancsok továbbra is helyesen hajtódtak végre, de ez hosszú távon nem volt használható. Szoftveresen sikerült a hibaarányt lecsökkenteni, de még így sem volt az igazi.



1.2. ábra: Első áramkör



1.3. ábra: Az első áramkör tervrajza

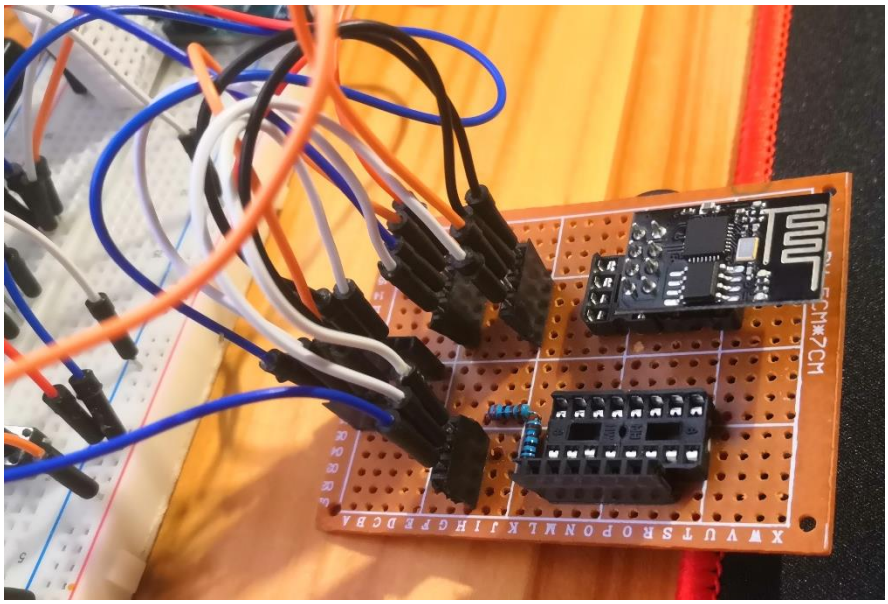
2 Kommunikáció a két eszköz között

2.1 Kapcsolat létesítése

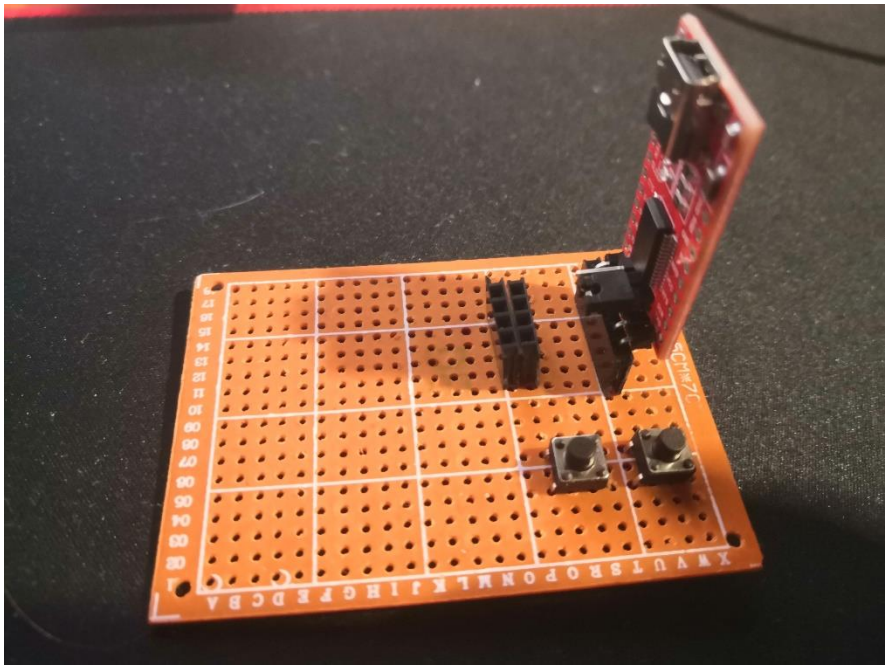
Az ESP8266-hoz van gyári firmware, ami úgynevezett AT parancsokkal történő kommunikációt tesz lehetővé. Az Arduino-n keresztül ezeket a parancsokat beírva lehet például hálózathoz kapcsolódni. Sajnos ezek nem mind vannak implementálva, és nagy különbségek vannak a verziók között, emiatt saját firmware-re lesz szükség. A gyári firmware-el az is a baj, hogy a telepítéséhez egy nagyon fapados download tool-t kell használni, amihez semmilyen dokumentáció nincsen (vagy ha van, az is kínaiul), így erre a megoldásra bármi komolyabbat tervezni ezért sem célszerű. A saját firmware írása nehezebb feladat, de jelenleg úgy tűnik, hogy jobban megéri a testreszabhatóság hosszabb távon. Ehhez szerencsére rendelkezésre áll az ESP8266Wifi[1] könyvtár, amit az Arduino IDE-be beimportálva egyszerűvé teszi a kód fordítását az ESP-re. A jelenlegi áramkörrel az a probléma, hogy nem igazán lehet új firmware-t tenni az ESP-re, mindössze az AT parancsokat engedi használni. Ezt az Arduino-t hídként használva meg lehetne oldani, de ennek használata nem praktikus – részben azért, mert akkor nem lehet magát az Arduino-t programozni közben – úgyhogy maradtam a külön eszköznél.

Mivel eddig minden csatlakozást forrasztásmentes breadboard-on csináltam, nem volt kényelmes a folyamatos kötögetés minden egyes kódfeltöltésnél, meg nem is túl jók a breadboard csatlakozásai, néha nem érintkezik minden rendesen. Úgyhogy felforrasztottam egy két hobbi üres nyáklapra egy két alap programozó áramkört. Az első próbálkozás (2.1. ábra) csak az ESP8266 lábait vezeti ki bedugható csatlakozókra, valamit van rajta több föld és tápfeszültség csatlakozó is. A második (2.2. ábra) már egy kicsit komolyabbra sikerült, nyomógombokkal ellátott, hogy a programozáshoz szükséges feszültségeket könnyen, huzalozás nélkül be lehessen állítani. Ezen az FTDI breakout-al lehet az ESP-t programozni, vagy kommunikálni vele.

[1] https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WiFi



2.1. ábra: 1. generációs ESP programozó áramkör



2.2. ábra: 1. generációs ESP programozó áramkör

2.2 Kommunikáció hibakeresés

Azért ez is elég távol áll a tökéletestől. Miközben ezekkel kísérleteztem, fény derült az egyik hibára, amit az eddigiek során elkövettem. Mivel az ESP 3.3 volton működik, az Arduino pedig 5 volton, ezért egy lineáris feszültség regulátort (LM317T), alkalmaztam a tápfeszültség konvertálására, a soros kommunikációra használt vonalra pedig egy egyszerű feszültségosztót tettem ellenállásokból. A hiba az volt, hogy a feszültség

regulátor be és kimenetére nem kapcsoltam kondenzátorokat, így a feszültség ingadozott, ebből adódtak az olvasási hibák. Ezek megoldódtak. Viszont lettek helyette mások.

Az ESP működés közben akár 180mA-t is fogyaszthat, ez az Arduino-val együtt már egy kicsit sok volt a gép USB portjának, ezért beruháztam egy külső táppal rendelkező USB elosztóra, ami akár több ampert is tud szolgáltatni. Később kiderült, hogy nem ez volt a probléma, de egyébként is jól jött pár extra port.

Sajnos az ESP nagyon szeret bekapcsoláskor – és néha máskor is – szemetet küldözgetni a soros portján. Amíg a számítógépről, manuálisan vannak kezelve az adatok, addig ez nem okoz problémát, viszont egy kommunikációra épülő, automata rendszernél ez nem kifejezetten előnyös. Így elkészítettem a kódokat, amiket az Atmega-ra és ESP-re fordíthatunk. Ezek már támogatnak hibakezelést is, tehát hibás üzenet, időtúllépés, vagy az input puffer túltöltése is kezelve van. Mindez tekintettel arra, hogy valós környezetben kicsit eltérően viselkedhet az eszköz, a kommunikációs linkek nem mindig tökéletesek.

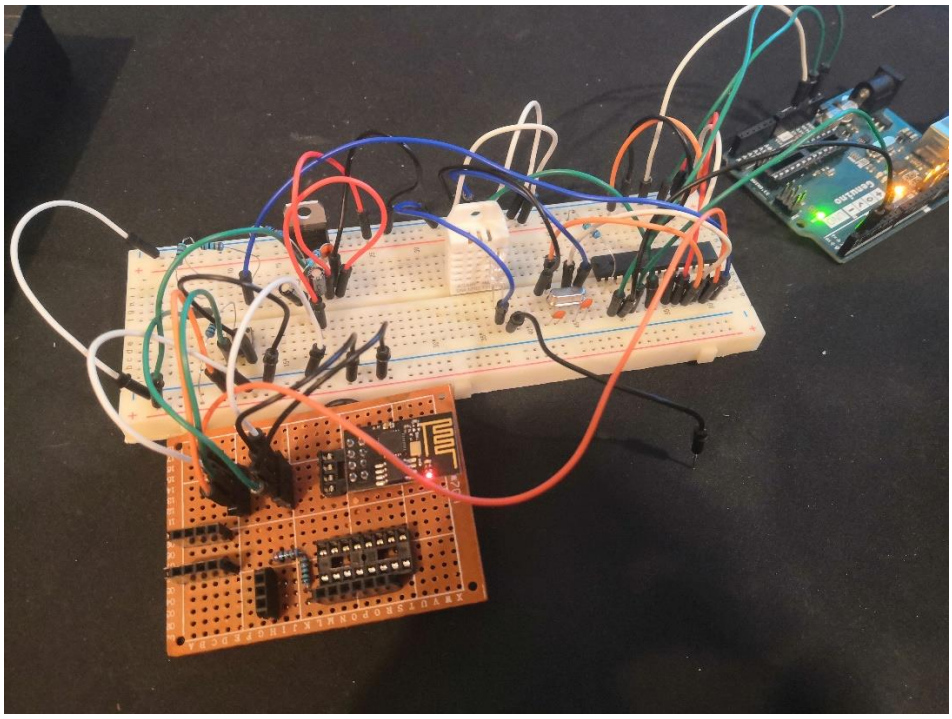
3 Az alapvető funkciók megvalósítása

3.1 Az eszközök programozása

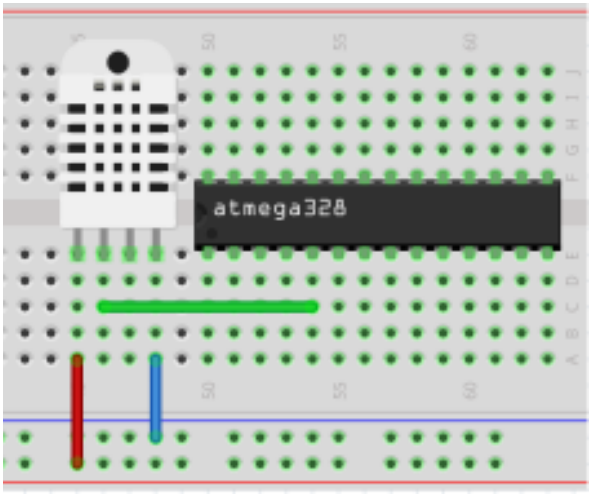
Miután ezek a kommunikációs problémák megoldódtak, az ESP firmware egy kiinduló állapota is elkészült. Ehhez ezt az internetes útmutatót[5] követtem. Gyakorlatilag azt csinálja, hogy csatlakozik a megadott WiFi hozzáférési pontok közül az épp legerősebbre, majd az egyik előre felkonfigurált pin-jén egy maximum 64 karakteres sztringet vár, c stílusút, ASCII 0-val végjelezve. Ezt a sztringet ezután elérhetővé teszi egy weboldalon, ami az ESP IP címét beírva érhető el. Itt az Arduino ide beépített serial monitor kommunikációs szoftvere kevésnek bizonyult, mivel nem lehet belőle sehogy escape-elni, nem lehet nem nyomtatható karaktereket küldeni. Ezért letöltöttem a Termite[6] programot, és erre a HexView plugin-t, amivel hexadecimális karakterkel bármilyen karaktert küldhető és fogadható, továbbá láthatóak a nem nyomtatható karakterek is. Ezt felhasználva az FTDI breakout-on keresztül lehet szöveget küldeni, amit az ESP elérhetővé tesz WiFi-n keresztül, böngészőből ellenőrizhető is a működése. Tényleg működik.

3.2 Szenzor bekötése

Következőnek a DHT22-ről való hőmérséklet és páratartalom adatokat kell az ESP-re juttatni, hogy ezt ki lehessen jelezni a webszerveren. Ehhez a DHT szenzor[7] könyvárat használtam az Adafruit-tól, ami az Arduino ide beépített könyvtárkeresőjében is elérhető és letölthető. Ezután az Arduino-ra rákötve pár sor kóddal hozzá lehet jutni az adatokhoz. Ezután az Arduino-ról ezeket az adatokat az ESP-re kell juttatni, amit soros kommunikáción keresztül valósítottam meg. Ehhez mindkét eszköz BAUD rátáját 9600-ra állítottam, mert kommunikációhoz ez se nem túl gyors, se nem túl lassú. Majd a lebegőpontos mért értékeket szöveggé alakítva egy kis kábelezés után el is tudtam küldeni. Ezek szintén megjelentek a böngészőben.



3.1. ábra: Működő áramkör szenzorral



1. ábra: A szenzor bekötésének tervrajza

Ez az ábra a jelenlegi állapotról. Itt viszont sajnos még voltak problémák a kommunikáció konzisztenciájával, csak néha küldött az ESP adatokat, az estek nagy részében csak hibaüzenetet küldött. Ezt az ATmega az új kóddal le tudta kezelni és addig indítgatta újra az ESP-t, amíg jó nem lett. Ezzel az a baj, hogy sokszor perceket vett igénybe. Amikor multiméterrel megmértem a feszültségeket, láttam, hogy az ESP-hez mindössze olyan 2.1V feszültség jut csak, ami nagyon kevés a 3.0-3.6 tartományhoz képest, amit a specifikáció leír. Ennek két oka is van. Az egyik, hogy az Arduino-n keresztül az USB portról csak olyan ~4.3V jött, ami lényegesen kevesebb, mint az 5, valamint a feszültség regulátoron is legalább 2V esik. Az egyik megoldás az volt, hogy másik áramforrásról

üzemeltettem a rendszert, amit 5.3V-ra állítva a működés megfelelőnek bizonyult, a hibák szinte teljesen eltűntek. A probléma az, hogy így nem az igazi a tesztelés, mivel azt USB-ről kell. A probléma úgy oldódna meg igazából, ha a feszültség regulátor helyett valami erre alkalmasabb eszköz kerülne beszerelésre. Ez majd még később más miatt is jó lesz. Viszont ehhez több alkatrész kell.

4 Működés akkumulátorról

A következő lépés az ATmega328P kiemelése az Arduino-ból. Ez azért jó, mivel ez az eszköz magában sokkal olcsóbb, mint egy egész UNO, valamint sokkal kisebb helyet is foglal, ami egy ilyen projektnél előnyös. Ez azért is fontos, mert a terv az, hogy az egész rendszert egy dobozban lehessen hordozni. Ehhez szükség van egy órakristályra, mert az ATmega alapból csak egy 8mhz-es van beépítve, ez nem valami pontos, továbbá kicsit nehézkes a használata. Ahhoz, hogy az ATmega-t ugyanúgy lehessen programozni, mint egy Arduino-t, először rá kell égetni az Arduino bootloadert. Ezt például egy UNO és egy kis kábelezés segítségével megtehető. Én ezt az útmutatót[8] követtem.

Az akkumulátorról való működéshez több minden is szükséges. Először is, az alacsony feszültség problémáját kellett megoldani, mert a sok újraindítás pazarolja az áramot. Ami itt igazán fontos, hogy ha a WiFi modul nem lesz mindig üzemben, hanem mondjuk csak 10 percenként egyszer, ami bőven elég releváns és pontos adatok kiolvasáshoz, akkor nagyon lecsökkenthető az áramfogyasztást, mivel a teljesítmény döntő részét a WiFi használata teszi ki. Igen ám, de ilyenkor csak akkor lehet az adatokat lekérdezni az eszközről, amikor az éppen fut, ami nem túl kényelmes, úgyhogy erre kell valami más megoldás.

4.1 Köztes szerver

Azt találtam ki, hogy egy köztes szervert üzemeltetek be – jelen esetben egy Banana PI M64, ami valahol áramforrásra és a hálózatra is csatlakoztatva van – és az ESP erre küldi majd az adatokat és innen lehet majd azokat kiolvasni, mert ez mindig futhat. Az Arduino és az ESP ezalatt mély alvásban lehet, hogy minél energiatakarékosabb legyen a rendszer. Erre a célra a Node-RED[9] platformot találtam leginkább alkalmasnak. Ennek segítségével folyamokat lehet létrehozni, amiken egy úgynevezett token utazik. Ezt úgy kell értelmezni, hogy minden beérkezett üzenet végigmegy az elemekből épített pályán, közben változásokon megy keresztül, hogy a végére a kívánt eredmény jelenjen meg. Ezeket viszonylag egyszerűen lehet összerakni – főleg az összehúzogatós felhasználói felületnek köszönhetően - de bármilyen javascript kódot is le tud futtatni az adatunkkal, úgyhogy szinte bármit meg lehet benne csinálni.

Ezen a felületen összeraktam egy szerver implementációt, ami egy JSON objektumot vár a megadott webcíment az ESP-től egy POST http kérésben, valamint azt egy UNIX időbélyeggel ellátva kiszolgálja egy másik címen, és egy naplóba is lementi. Így innen bármikor lekérdezhetjük az adatokat, és az időbélyegnek köszönhetően azt is tudjuk, mikoriak, így ha esetleg lemerült az elem, vagy valami hiba lépett fel, akkor ezt tudjuk orvosolni.

4.2 Kommunikációs protokoll

Az eddigi fejlesztésekkel kezdett az ESP és az Arduino kódja is egyre bonyolultabbá válni. Ezek kezdetben csak demonstrációs kódok voltak, és egyre nehezebb volt rajtuk módosításokat végezni, úgyhogy újra írásra volt szükség, valami olyanra, ami könnyen módosítható, és hatékonyan használja ki a hardver erőforrásait. Egy olyan protokollt találtam ki, ahol mindkét eszköznek van egy adott méretű utasítás és üzenet puffere. Alapértelmezésképpen mindketten az utasításpufferbe olvasnak, de egy utasítással meg lehet nekik mondani, hogy a hosszabb üzenet következik, és használják a másikat. Ezzel hatékonyan kezelhetők a kommunikációs hibák és új funkcionalitás bevezetése is könnyen elintézhető.

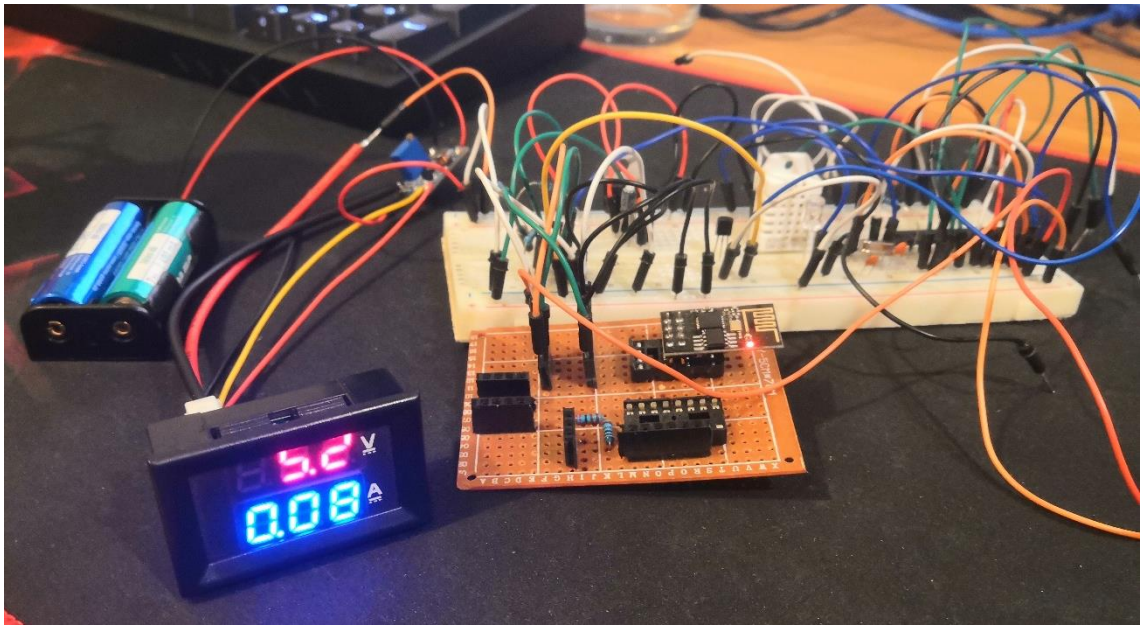
A kommunikációs folyamat a két eszköz között nagyjából úgy zajlik le, hogy az Arduino elküldi a mért adatokat speciális karakterekkel elválasztva, majd ezt az ESP egy JSON objektummá alakítva feltölti a szerverre. A kommunikációba több kézfogás is be van építve, hogy minél kisebb eséllyel alakuljon ki livelock vagy deadlock a rendszerben, mert az az akkumulátor gyors lemerülését eredményezi.

5 Fizikai implementáció

5.1 Áramellátás optimalizálása

Így már el lehet kezdeni beüzemelni az akkuról működő verzióját a projektnek. Ehhez 2db 2500mAh kapacitású AA-s NiMH cellát használtam, de alkáli elemmel is működik. Az akkuk feszültsége valamennyire függ a töltöttségi szintről, a két cella kombiált ~2.5V-os feszültsége egyéként is kevés, úgyhogy itt egy boost converter-re van szükség, amivel fel lehet konvertálni a feszültséget egy többé kevésbé konstans 5V-ra. Nekem szerencsére volt itthon ilyenből egy pár, amit még eBay-ről rendeltem. Így már tudtam elég feszültséget biztosítani az áramkörnek, ami gond nélkül működött.

Ezek után rákötöttem egy feszültség és árammérőt a tápellátásra, hogy megmérjem a fogyasztást, hogy ennek segítségével meg tudjam becsülni az üzemidőt. Erről az alábbi képet készítettem.



5.1. ábra: Fogyasztás mérése

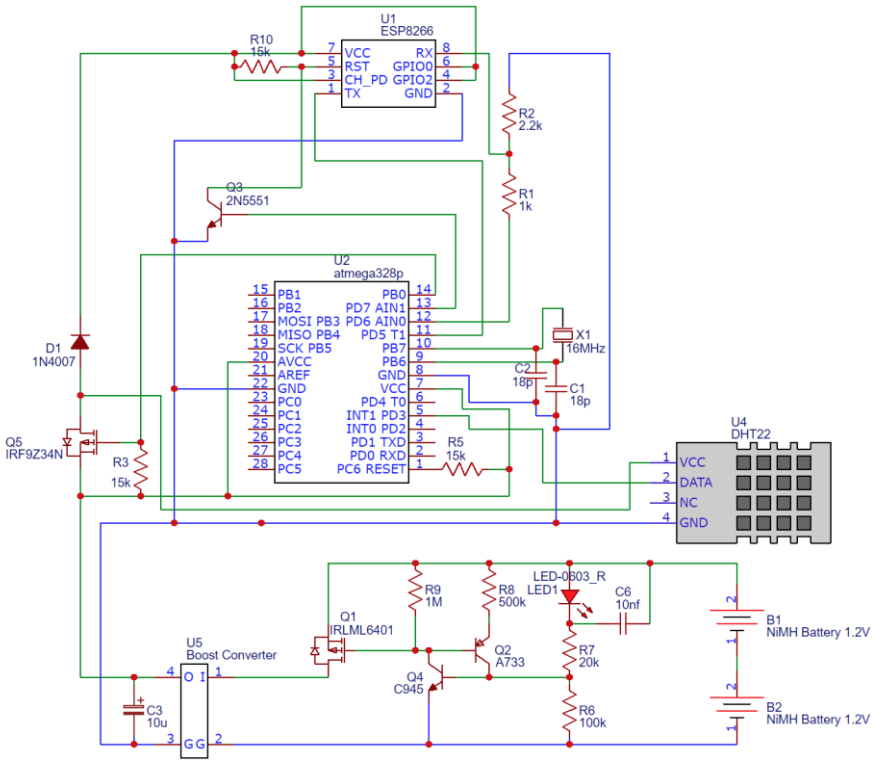
Ez alapján olyan 80mA-t az áramkör, amikor a wifi be van kapcsolva. Amikor nincs, akkor ez a mérő 0-t mutatott, ami sajnos nem olyan jó hír, mint aminek tűnik, mivel a mérő viszonylag pontatlan, úgyhogy 15-20mA alatti értékek mérésére nem alkalmas. Megpróbáltam a multiméteremmel is megmérni az áramfogyasztást, ez elég inkonzisztensnek bizonyult, csak akkor tudott az áramkör egyáltalán bekapcsolni, hogy megemeltem a feszültséget majdnem 0.5V-al. Ez arra utal, hogy a multiméter érintkezései

az áramkörrel elég rosszak, ami nem meglepő, hiszen csak a kezemmel tartottam őket a fém kontaktokhoz. Így 18mA fogyasztást mértem alvás alatt, de ez a szám valószínűleg nem túl pontos. Amit észrevettem, hogy ha kihúzom az ESP-t akkor a fogyasztás lecsökken 1.8mA-ra, ami egy sokkal vonzóbb érték, de ez sem pontos, ugyebár. Azt találtam ki, hogy a boost converter célfeszültségét leveszem kb 4.8V-ra, a feszültség regulátort lecserélem egy FET-re, valamint rákötök egy diódát a feszültség lejjebb vitele miatt. Ezzel sikerült a gyakorlatban is elérni az 1.8mA-s mért értéket, úgy, hogy nem húztam ki közben semmit. Ezekkel a feszültségértékekkel számolva, úgy, hogy a rendszer 10 percenként frissít, és egy frissítés átlagosan 15mp-is tart, megbecsülhetjük a várt akkumulátor üzemidőt. Erre ~300 óra jött ki, ami kicsit kevesebb, mint két hét. Ez már egész jónak tűnt, bár ezek olyan becslések, amiknek csak a nagyságrendjét érdemes figyelembe venni.

Itt eszembe jutott, hogy az áramforrás túlmerítése ellen is kell kezdeni valamit. Mivel ez egy olyan eszköz, amit nagyrészt magára hagyunk, ezért nem lenne szerencsés, ha hagynánk kifolyni az akkukat, elemeket. Ennek az internetes útmutatónak[10] alapján megterveztem egy kétcellás merítő áramkört, amiben hiszterézis is van, 1.88V körül kapcsol ki, és 1.94V körül be, így megakadályozhatjuk, hogy a merülés végén az áramkörünk össze-vissza kapcsolgasson ki és be. Ennek a kapcsolási rajzát az EasyEDA[2] ingyenes online szoftverrel készítettem el. Ugyanide felvittem az eddig elkészített projekt többi részét is, mert szeretném, ha nem csak emlékezetből lehetne reprodukálni az egészet. Továbbá ez ahhoz is kiemelkedően hasznos, hogy fel tudjam forrasztani az alkatrészeket egy prototípus nyáklapra.

[2] https://easyeda.com/

5.2 Forrasztás

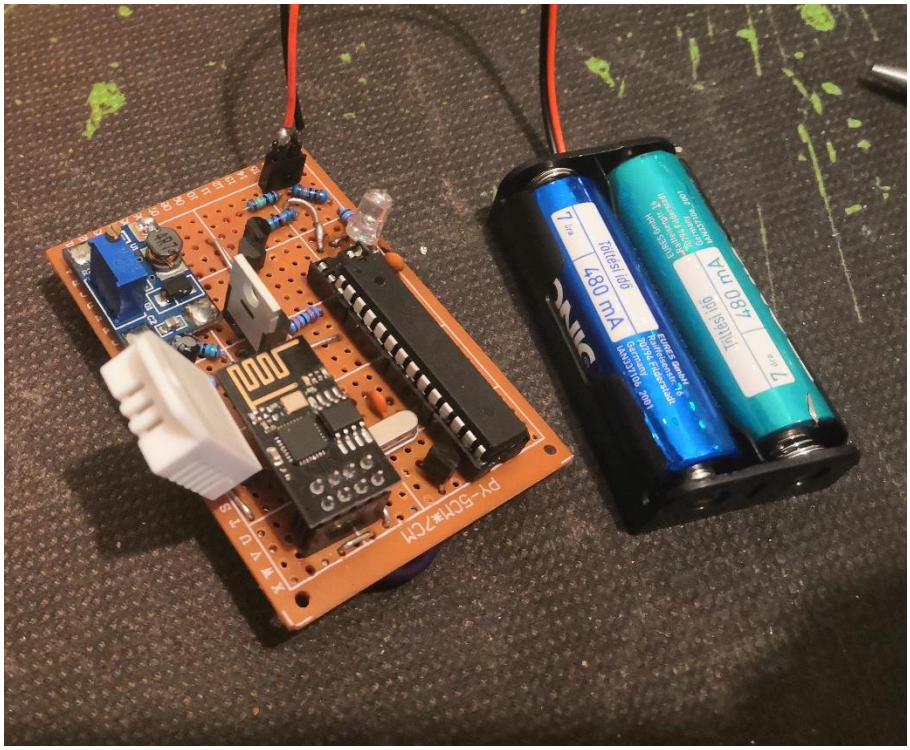


5.2. ábra: A mérőeszköz kapcsolási rajza

Ezen az ábrán a projekt többé-kevésbé végleges kapcsolási rajza látható. A forrasztást ennek alapján készítettem. Illetve, ez így nem teljesen igaz, az eredeti rajzon véletlenül összecseréltem az ESP RX és TX pin-jét, így a két eszköz nem tudott kommunikálni, valamint a 3.3V-os ESP-nek nem nagyon tetszett a fogadó vonalára rakott 5 volt, el is kezdett hevülni. Szerencsére viszonylag gyorsan észrevettem, ezért túlélte. Egy két hiba azért esett a forrasztás közben például, hogy a feszültségosztót rosszul kötöttem be, de nagyrészt gördülékenyen ment.

Itt implementáltam először az akkumulátor védelmi áramkört. Megpróbáltam breadboard-on is, de mivel a kapcsoláshoz használt MOSFET kizárólag SMD csomagban létezik, ezért amikor a lábaira forrasztottam a csatlakozásokat, letörtem őket sajnos. Ennek a felforrasztás közben egy rövid zárat kivéve – amit gyorsan elhárítottam – minden simán ment. Legalábbis egyelőre úgy tűnt. Mivel nincs változtatható feszültségű tápegységem, ezért nem tudtam tesztelni, hogy a helyes értéknél zár-e a védő áramkör. Minden jól működni látszott, az eszköz a vártnak megfelelően küldte az adatokat a szerverre, azokat le tudtam olvasni. Ekkor felmentem AliExpress-re, hogy alkatrészeket

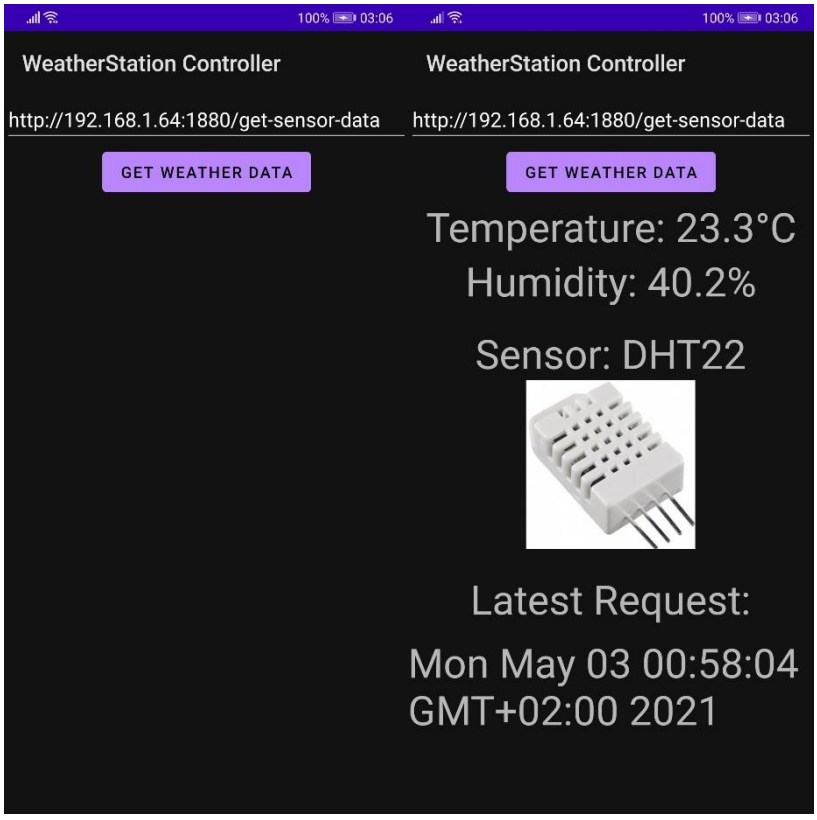
vásároljak a további projektjeimhez. Itt olyan 1V-os zener diódát kerestem, amit a védő áramkör leírásában említve láttam, de nem találtam. Itt elkezdtem google-ön keresni, hogy hol lehet ilyet kapni, és ráakadtam egy fórumra, ahol pont arról a Conrad-ból származó zener diódáról beszéltek, amit én a projektemhez beszereztem. Itt azt olvastam, hogy valami elírás lehetett az adatlapon, mert ilyen paraméterekkel dióda nem nagyon létezik. Itt előkaptam megint a pákát, és gyorsan kicseréltem ezt egy piros led-re. Ez a led kizárólag a feszültség karakterisztikája miatt van itt, fényt szinte nem is ad ki, így a fogyasztásával sem lesz probléma. Legalábbis ezt gondoltam, egy kis tesztelés gyorsan felfedte, hogy a led körülbelül egy nap alatt lemeríti az akkumulátorokat, ami igen kevés, mint üzemidő. Szerencsére így működött a túlmerítés védelem, de ennek ellenére ez még újra tervezésre szorul. Itt egy kép a kész áramkörről.



5.3. ábra: Az elkészült mérőeszköz, akkumulátorra kötve

6 Mobilalkalmazás

A projekthez alkalmazást is fejlesztettem Android platformra. Ennek a célja, hogy a szerverről lekérdezve ne egy nyers JSON sztringet kelljen nézegetni a böngészőben, hanem egy felhasználóbarát felületről lehessen ezeket leolvasni. A UNIX időbélyeget olvasgatni sem kifejezetten kellemes. Így az Android Studio-ban létrehoztam egy egyszerű Activity-t, amiben vizuális elemeket helyeztem el. A http kéréseket az OkHttp[11] könyvtárral oldottam meg, mert ez könnyű aszinkron kezelést biztosít. A szerverről lekért JSON objektumokat pedig a Moshi[12] könyvtárral készítettem. Ennek az az előnye, hogy kevés kódot kell írni, valamint fordítási időben generálja a kódot, ami azt jelenti, hogy egyáltalán nem lassítja a programot.



6.1 ábra: A mobilalkalmazás

Így néz ki a program. A bal oldali kép látható az alkalmazás megnyitásakor a jobboldali, meg miután a helyes webcím beírása után megnyomásra került a gomb. Az alkalmazás kiírja a hőmérsékletet és a páratartalmat, a szenzor típusát (ezekhez az alkalmazás képeket is rendel, de ezt minden szenzorra külön bele kell tenni) és hogy mikor készült az utolsó mérés.

7 Akkumulátoros működés technológiai fejlesztése

7.1 NiMH technológiák

Az eszköz eddig NiMH, vagyis Nikkel-metál-hidrid, akkumulátorról működött saját készítésű merítő áramkörrel, ennek előnyei és hátrányai is voltak. Talán a legnagyobb előny, hogy ezek az akkumulátorok igen széles körben elérhetőek, valamint, hogy kompatibilis AA-s elemmel is, így lemerülés esetén nem kell a töltésre várni, hanem egyből, az akkuk/elemek cseréje után azonnal használható. A fő hátrány az én véleményem szerint, hogy a NiHM akkukkal felszerelt készülék nem tölthető magában (pl. USB-n keresztül), mivel ezeknek az akkuknak speciális töltőre van szüksége, amit nem praktikus a mérőeszközre integrálni. Ez a napelemes működést is kizárja.

7.2 Li-ion és Li-po technológiák

A lítium alapú akkumulátor technológiák manapság már a leginkább elterjedtek, mobiltelefonokban és elektromos autókban is ilyenekkel találkozunk. Legfőbb előnyük, hogy nagyon nagy az energiasűrűségük, tömegarányosan akár 3-szorosa a NiMH technológiának[13]. Ez ebben a projektben nem olyan kritikus, mivel a készülék, amit árammal akarunk ellátni alacsony fogyasztású. Ezen felül a töltésük is sokkal egyszerűbb, 5V-os feszültségre kapcsolva tölthetők, nekünk csak arra kell figyelni, hogy ne töltsük túl, illetve ne zárjuk rövidre őket. Ezek könnyen és olcsón megoldhatók például egy előre elkészített, erre specializált áramkörrel.

7.1. ábra: Li-Ion akkumulátor

7.3 Li-ion és Li-po közti különbségek

A két technológia igen hasonló, mind kémiai összetételben, mind használatban. A legfőbb különbség, hogy különböző elektrolit található az anód és a katód között. Ennek köszönhetően a Li-Po akkumulátorok szinte bármilyen méretben és formában megtalálhatóak, míg a Li-Ion akkukkal általtában szabványos méretű, henger alakú tokban találkozhatunk.

7.2. ábra: Li-Po akkumulátor

8 Szerver backend technológiák

8.1 Választott technológiák

Az első prototípusban használt Node-Red keretrendszer nem bizonyult elégségesnek a projekt számára. A cserére azért volt szükség, mert ilyen előre gyártott rendszereknél kevés a testreszabhatóság, vagy ha van is, akkor egy idő után nagyon nehezen kezelhető a komplexitás miatt. A választásom a Fiber backed keretrendszerre esett, ami a Go programozási nyelvben íródott. Ezt mindet egy konténerizált környezetben valósítottam meg, docker segítségével.

8.2 Docker felépítés

A rendszer dinamikus használatához a „docker compose”-t használtam, amiben három konténer fut. Egy MariaDB konténer, ez az adatbázis, ami tárolja majd a felhasználókat és a mért szenzoradatokat. Egy PhpMyAdmin konténer, ez nem elengedhetetlen a működéshez, de a karbantartáshoz és főként a fejlesztés közben hibakereséshez igen hasznos. És végül maga a Fiber konténer, ami magát a backendet tartalmazza, ehhez érkeznek be a felhasználótól és az állomástól a kérések, majd feldolgozás és esetleges adatbázis műveletek után választ küld.

8.3 A backend rendszer fő funkciói

A rendszer rengeteg különböző funkciót lát el, hogy a működés hatékony és biztonságos legyen. Ezek a funkciók úgy érhetőek el, hogy különböző helyekre JSON objektumokat kell küldeni, és erre a szerver küld egy sikeres vagy hibaüzenetet. Lekérdezések esetén pedig egy JSON objektumot küld vissza, amiben a kért adatok vannak. Ezek a funkciók:

8.3.1 Felhasználó regisztrálása

Minden felhasználónak regisztrálnia kell, mielőtt használhatja a rendszert. Erre azért van szükség, hogy több, egymástól független szenzor klasztert is kezelni lehessen egy szerverről. A regisztrációnál meg kell adni egy felhasználónevet és egy jelszót. A felhasználónévnek egyedinek kell lennie. Ha sikeres a regisztráció, akkor egy egyedi numerikus kulcsot rak hozzá az adatbázis. A jelszavakat nem, csak a jelszó hash-e kerül mentésre, egy salt hozzáfűzése után. Ez kiemelkedően fontos biztonsági lépés, mert így ha az adatokat el is lopják, a jelszavakat ilyenkor sem fogják tudni visszafejteni, ha erős

jelszót választottunk. Ezentúl minden további műveletnél a kérésbe a felhasználónevet és a jelszót is bele kell rakni, mert lesz hitelesítve a művelet, és biztosítva, hogy mindenki csak a saját adataihoz fér hozzá.

8.3.2 Szenzor regisztrálása

Minden felhasználó szenzorokat tud regisztrálni, ezeknek a mérései kerülnek majd az adatbázisba. Szenzor regisztrációjánál nevet kell adni a szenzornak. Ennek felhasználónként egyedinek kell lennie, mert ezt azonosítóként használjuk. Itt azért nem célszerű numerikus azonosítót bevezetni, mert akkor a felhasználó nem fogja tudni, hogy éppen melyik szenzort nézi.

8.3.3 Szenzoradatok feltöltése

Ezt a feladatot maga a szenzoros egység végzi. Az adatbázis minden feltöltött adathoz időbélyeget csatol a létrehozás időpillanatában. Maguk a mért adatok JSON sztring-ként érkeznek és az adatbázisban egy szöveges mezőben vannak eltárolva. Ez azért előnyös, mert ide bármilyen adatok kerülhetnek, vagyis ha többféle szenzorunk van, lehet például egy olyan, ami nyomást is képes érzékelni, akkor nem kell semmilyen módosítást végezzünk se az adatbázison, se a backend rendszeren, hanem a mért adatok hibátlanul meg fognak jelenni a felhasználói interfészen.

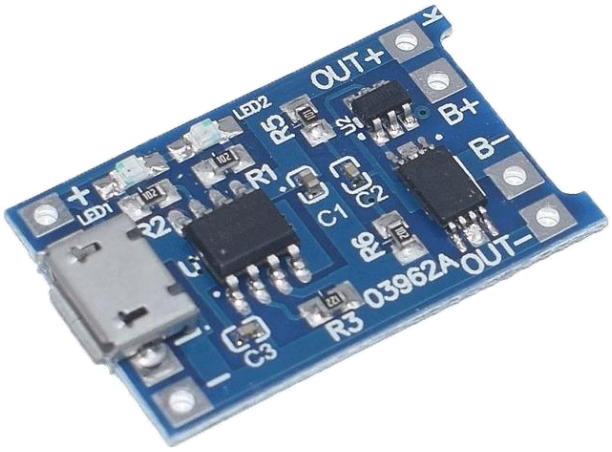
8.3.4 Szenzoradatok lekérdezése

Az adatok lekérdezéséhez meg kell adni egy szenzort és egyidőbélyeget, a szerver pedig az ehhez a szenzorhoz tartozó, az időbélyegnél frissebb adatokat fogja elküldeni. Ezt limitálni is lehet, hogy ne terheljük túl az adatbázist, ha valaki véletlenül több tízezer rekordot akarna lekérni.

9 Áramkör módosítása

9.1 Akkumulátor technológia váltása

Az áramkörből kikerült a NiMH merítő rész, mivel ezentúl kizárólag lítiumos akkumulátorral lesz kompatibilis. Ehhez rákerül helyette egy előre gyártott Li-Ion töltő és merítő modul, ami a TP4056 integrált áramkörön alapul. Ez a chip szabályozza a Li-Ion cellák töltését, merítését annak érdekében, hogy az élettartamuk minél hosszabb legyen. Ezen felül többféle védelmet is nyújt, például rövidzár ellen is. Így, ha valamilyen oknál fogva túl nagy áramot kellene az akkumulátornak leadnia, akkor azonnal lekapcsolja a kimenetről, megvédve ezzel a sérülékeny cellát.



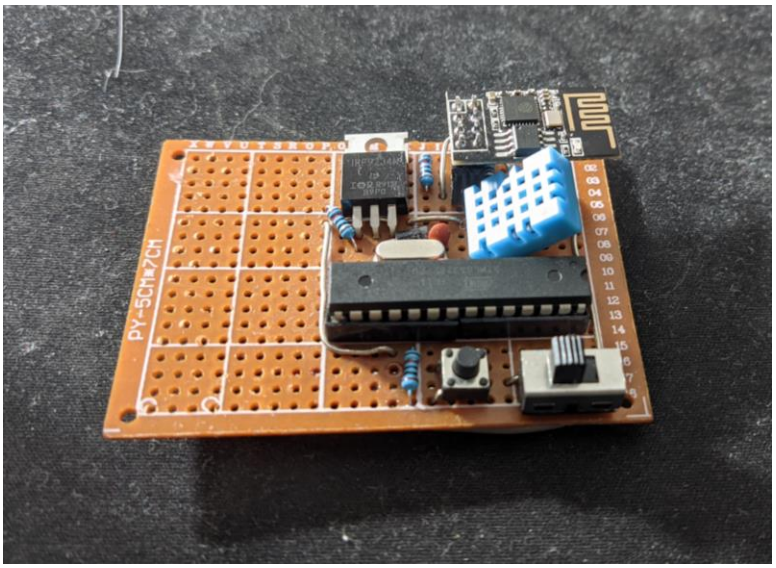
9.1 ábra: A töltő, merítő és védő áramkör

9.2 Nyomtatott áramkör tervezése

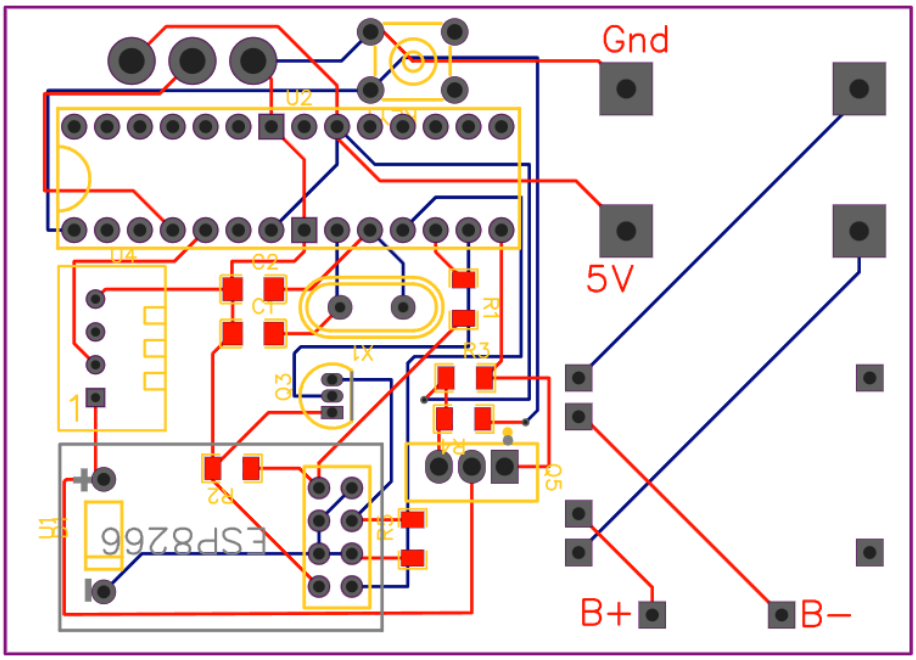
Az első modell kizárólag a kapcsolási rajz alapján lett nyáklapon elkészítve, az alkatrészek és vezetékek tényleges elhelyezkedését a forrasztás közben találtam ki. Ez egy ilyen kis projektnél még megtehető, de nagyobbaknál már nem járható. Ezért készítettem, az EasyEDA web program segítségével egy dizájnt, amin már minden méret és kapcsolás fel van tüntetve. Ez azért is előnyős, mert ha rendelkezünk egy ilyen úgynevezett PCB Fabrication File (vagy Gerber file) -al, akkor ez megrendelhető egy profi PCB gyártó cégtől. Ilyenkor nem kell az alkatrészek közti kapcsolásokkal foglalkozni már a forrasztásnál, sőt, az egyszerűbb alkatrészeket, mint például ellenállások, kondenzátorok, de akár gyakoribb IC-k is, szintén felrakathatók a gyártóval. Ráadásul, ha így döntünk, akkor a régi stílusú, leginkább hobbiból használt THT (Through Hole Technology, vagyis Furatszerelési technológia) alkatrészek helyett

használhatjuk ezeknek az SMD (Surface Mount Device, vagyis Felületszerelt készülék) ekvivalensét, amik sokkal kisebbek és olcsóbbak, valamint a nyáklapnak csak az egyik oldalán helyezkednek el, ezzel is több helyet spórolva.

A Gerber file az eredeti tervekhez képest kiegészült egy reset (újraindítás) gombbal, egy setup (beállítás) mód kapcsolóval, valamint furatokkal az előre gyártott áramkörökkel való kapcsolatoknak. Ez előbbiek maga az érzékelő eszköz működéséhez szükségesek.



9.1. ábra: Áramkör az új tervek szerint



9.2. ábra: Az új áramkör digitális tervrajza

10 Az eszköz szoftveres működése

10.1 Funkcióbéli változások

A két eszköz kommunikációja sok változtatást igényelt, legfőképpen praktikusság szempontjából. Eddig a WiFi hálózat SSID-ját és jelszavát, valamint az eszköz felébredésének frekvenciáját csak a forráskód módosításával, lefordításával és újból feltöltésével lehetett módosítani. Ez azért kényelmetlen, mert lehet, hogy ezeket gyakran is akarjuk változtatni. Erre dolgoztam ki egy új módszert, hogy mindezt egyszerűen, kizárólag egy okostelefon segítségével lehessen kezelni.

Először is szükség van arra, hogy dinamikus adatokat is tároljunk az ESP és az Atmega tárhelyén. Eddig ezzel nem kellett foglalkozni, ezeket mind magában a kódban tároltuk. Szerencsére mindkét mikrokontroller rendelkezik flash memóriával, amit írni és olvasni tudunk, valamint az eszköz kikapcsolása és újraindítása után is megmarad. Ez utóbbi azért nagyon fontos, mert nem lenne kényelmes minden egyes lemerülés vagy újraindítás után újra bekonfigurálni az eszközünket.

10.2 Adatok feltöltése

Ahhoz, hogy az adatokat fel tudjuk tölteni az eszközünkre, teljesen más működési módban kell elindítani azt. Nem tudjuk a köztes szerveren keresztül átvinni az adatokat, mivel a WiFi jelszót nem tudja az eszközünk, hiszen épp azt akarjuk beállítani. Erre megoldás a nyáklapon újonnan található gomb és kapcsoló. A kapcsoló átbillentésével és az eszköz újra indításával beállítás módban indíthatjuk el az eszközt. Ilyenkor ahelyett, hogy egy a környezetben lévő WiFi hálózathoz csatlakozna, létrehoz egy sajátot, amire például telefonnal csatlakozhatunk. Ha ez megtörtént, ezen a hálózaton a megfelelő címre küldött JSON objektummal módosíthatjuk a tárolt adatokat. Ezután visszabillenthetjük a kapcsolót és újra indíthatjuk az eszközt, ami ezután már az újonnan beállított konfiguráció szerint fog működni, egészen addig, amíg ezt újból megváltoztatjuk.

10.3 Szoftveres megvalósítás

Ezen előbbiek megvalósítása a jelenlegi hardveres konfiguráción nem egyértelmű. A főbb probléma az, hogy a WiFi SSID és jelszó az ESP-n kell, hogy legyen tárolva, hiszen ez a modul kapcsolódik a hálózathoz. A bekapcsolási intervallumnak viszont az Atmega-n

kell lennie, mert ez a kontroller szabályozza az egész eszköz ki- és bekapcsolási állapotát, az ESP-t és a szenzorokat olykor teljesen áramtalanítja a hosszabb üzemidő érdekében. Ezt a problémát úgy hidaltam át, hogy a már létező kommunikációs protokollt felhasználva a szükséges adatokat, miután azok megérkeztek ez ESP-re JSON formájában, átküldtem az Atmega-ra és ott a flash memóriában eltároltam. A maradékot meg már az ESP-n mentettem, ugyanilyen módon. Ezek után, amikor működés módban indítjuk el az eszközt, akkor mindkét mikrokontroller a saját tárhelyéről olvassa ki a szükséges adatokat, és ezeknek megfelelően működnek majd.

11 A szenzorok működése

A dolgozat egyik fő feladata, hogy több féle szenzorra is könnyen adaptálható legyen. Én a DTH11 és a DHT22 hő és páratartalom érzékelőket választottam, mint demonstráció. Ennek több oka is van. A legelső, annak köszönhetően, hogy az adatokat soros interfészen keresztül kérhetjük le, hogy ezeket a szenzor modulokat kifejezetten könnyű kezelni az Atmega328P mikrokontrollerről, amit a projektben használunk. Ezenfelül rendkívül olcsók, a DHT11 körülbelül 300 Ft áron már a miénk lehet, a DHT22, ami az előbbinek az erősebb változata, nagyjából 1000 Ft-os áron szerezhető be. A különbség a kettő között teljesítmény szempontjából, hogy a DHT22-es modul nagyobb intervallumban tudja mérni mind a hőmérsékletet, mind a páratartalmat, valamint pontosabban is. Ezek a méréshatárok a hőmérséklet esetén -40-80 °C, valamint 0-50 °C, illetve +-0.5 °C, valamint +-2 °C pontossággal, páratartalom esetén pedig 0-100%, valamint 20-90%, illetve +-2%, valamint +-5% pontossággal, ilyen sorrendben. Minkét modul 5V-os tápfeszültséget igényel, ez is előnyös, mivel az Atmega is 5V-ot használ, így nem kell a feszültség konvertálásával bajlódjunk.

11.1 Kapcsolás a mikrokontrollerhez

A nyák dizájnban, amit készítettem, egy DHT szenzornak van dedikált hely fenntartva, valamint ez rendelkezik a működéshez szükséges kapcsolásokkal, mint például a tápfeszültség ez az adatvonal, így mind a 11-es, mind a 22-es modul rászerelhető, az azonos lábnyomuknak köszönhetően. Ezenfelül más modulok kapcsolására is van lehetőség, például egy DS18B20+ hőérzékelő, vagy egy BMP280-M hőmérséklet, magasság és légnyomás érzékelő, sőt többet is egyszerre. Ez annak köszönhető, hogy az Atmega nagyon sok GPIO[3] lába van, ebből a példa áramkör, amit készítettem, csak kevesebb, mint felét használja, így több mint 6 általános felhasználású port még szabadon van, valamint még specifikus protokolloknak fenntartott lábak, például SPI[4]. A különböző lábnyom sem probléma, mivel a nyák dizájn nyílt, és könnyen módosítható, hogy másféle elemeket is elhelyezhessünk rajta.

[3] General Purpose Input/Output, Általános Célú Be- és Kimenet
[4] Serial Peripheral Interface, Soros Periféria Interfész

11.2 Szoftveres háttér

Mindez a hardveres kompatibilitás nem elég önmagában, az érzékelőktől érkezett adatokat valahogyan kezelni is kell. Szerencsére a dolgozat részeként tervezett és implementált kommunikációs protokoll bármilyen adat átvitelére alkalmas. Az átviteli puffer mérete egyszerűen, makrókkal van megvalósítva, így összesen egy számot kell a kódban átírnunk, ha több adat átvitelére van szükségünk, illetve csökkenthetjük is a méretet a hatékonyabb működés érdekében. A JSON formátumnak köszönhetően a különböző adattípusokat, például nyomás, hőmérséklet, sem kell külön kezeljünk, ezt a rendszer magától tudja interpretálni.

12 Szenzoradatok felhasználása

12.1 Alapvető felhasználás

Az adatok a mérőeszközről való feltöltés után a szerveren, egy MySQL kompatibilis adatbázisban kerülnek eltárolásra. Az SQL robusztussága lehetővé teszi az adatok sokféle lekérdezését és manipulációját. A legalapvetőbb felhasználása az adatoknak, hogy a felhasználó láthassa a legutóbb mért hőmérsékletet, ami, ha nem túl régen készült, tekinthető az aktuális hőmérsékletnek a szenzor helyén. Ez önmagában is hasznos tud lenni, de a felhasznált technológiák adta lehetőségekkel ennél jelentősen több információt tudunk kinyerni a rendelkezésre álló adatokból. Ennek egy alapvető módja, hogy az adatbázist felhasználva nem csak a legutóbbi, hanem egy adott időintervallumon belül az összes mért adatot láthatóvá tesszük a felhasználó számára. Ez megnyilvánulhat például egy gráf formájában, amin a hőmérséklet van illusztrálva az idő függvényében. Így már lehetőségünk van statisztikákat készíteni, meg tudjuk állapítani a hőmérséklet átlagát, szórását, mediánját, amelyek mind a való életben hasznos információval bírnak. Megállapíthatjuk belőle például a helyiség hőtartó képességének mértékét, és hogy célszerű-e szigetelni.

12.2 Adatok együttes alkalmazása

Ha egyszerre nem csak egyféle adatot, például a hőmérsékletet vesszük figyelembe, hanem különböző adattípusokat egyszerre elemzünk, akkor sokkal szélesebb képet kaphatunk az adott szituációról. A hőmérséklet és páratartalom közös értelmezésekor meg tudjuk állapítani a levegő vízgőzzel való telítettségét. Ez hasznos lehet például egy olyan környezetben, ahol sok számítógép vagy más elektronikai eszköz található, mivel itt a pára kicsapódása általában nem kívánatos. Használhatjuk ezt az információt arra, hogy mikor érdemes a légkondicionáló egységet bekapcsolni, vagy ha a lehetőségek engedik, egy költséghatékonyabb párásító hűtőt beszerezni. Több különböző pontban elhelyezett eszköz használatával pedig akár a helyiségek közti szellőzést, illetve hőátadást is mérhetjük, rengeteg lehetőség rejlik még itt.

12.3 Előrejelzések alkotása

Nem lehet a hőmérséklet és hasonló adatok méréséről beszélni anélkül, hogy az időjárás előre jelzésről is ejtenénk pár szót. Mint azt a mindenki számára elérhető előrejelzésekből láthatjuk, ez egy nehéz feladat. Ennek a fő oka az, hogy az időjárást meghatározó folyamatok turbulensek, ami azt jelenti, hogy nagyon kicsit bemeneti változásra is teljesen más eredményt kapunk. Így olyan általános következtetéseknél, mint hogy a légnyomás hirtelen csökkenése gyakran esőzést okoz, nem tudunk sokkal többet mondani, különösen olyan primitív eszközökkel, mint amik egy ilyen árkategóriájú eszköznél szóba jöhetnek. Kis előnyt nyújthat nekünk a neurális hálók, illetve a gépi tanulás alkalmazása, de a káosz okozta akadályokon ezzel sem tudunk felül kerekedni.

13 Értékelés

Úgy érzem, hogy jóval tovább sikerült eljutni az időjárásállomás fejlesztésével, mint azt eredetileg terveztem, de még így sem elég messzire. A feladat milyensége és a saját magam által kikötött elvek miatt egy hasznos funkció bevezetése két másikhoz nyitotta meg a lehetőséget. Ha most kellene folytatni, ezeket venném legelőre.

Fejleszteném a biztonságot. Ezen a területen nem lehet eleget fejleszteni. Az emberek online eltöltött ideje, valamint az online eszközök száma soha nem látott mértékben növekszik az elmúlt években, és ez csak gyorsulni látszik. Ezzel együtt miden internethasználó, egyre nagyobb fenyegetésnek van kitéve, mivel a támadóknak egyre inkább megéri erőforrásokat fektetni az adatlopásba. A rendszer védve van SQL injection, valamint puffer túlcsordulásos támadások ellen, amik a leggyakoribbak közé tartoznak. A legelső fejlesztésem az SSL titkosítás lenne, amivel titkos csatornán tudunk majd a szerverrel és a mérőeszközzel kommunikálni, valamint a MIM[5] támadások ellen is védelmet nyújt, de hátulütői is vannak.

Összességében, nagyon szerettem a projekten dolgozni, mindig felülülés volt a tradicionális egyetemi teendőköz képest. Örültem, hogy valami újat alkothattam, úgy, hogy közben minden percben a saját tudásomat gyarapítottam.

[5] Man in the Middle attack, Közbeékelődéses támadás

Irodalomjegyzék

[1] Dave Evans: *The Internet of Things: How the Next Evolution of the Internet Is Changing Everything*, *CISCO White Paper*, (April 2011)

[2] Alexander S. Gillis: *What is internet of things (IoT)?*, https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT (2021. dec.)

[3] argenox.com, *Ultimate Guide to Internet of Things (IoT) Connectivity*, https://www.argenox.com/library/iot/ultimate-guide-iot-connectivity/ (2021. dec.)

[4] Andrew Meola, *Smart Farming in 2020: How IoT sensors are creating a more efficient precision agriculture industry*, https://www.businessinsider.com/smart-farming-iot-agriculture (2021. dec.)

[5] Pieter P.: *A Beginner's Guide to the ESP8266*, https://tttapa.github.io/ESP8266/Chap01%20-%20ESP8266.html (2021. dec.)

[6] CompuPhase: *Termite*, https://www.compuphase.com/software_termite.htm (2021. dec.)

[7] adafruit: *DHT-sensor-library*, https://github.com/adafruit/DHT-sensor-library (2021. dec.)

[8] Arduino: *From Arduino to a Microcontroller on a Breadboard*, https://www.arduino.cc/en/Tutorial/BuiltInExamples/ArduinoToBreadboard (2021. dec.)

[9] OpenJS Foundation & Contributors: *Node-RED*, https://nodered.org/ (2021. dec.)

[10] tfried: *2 Cell NiMH Battery Protection Circuit(s)*, https://www.instructables.com/2-Cell-NiMH-Battery-Protection-Circuits/ (2021. dec.)

[11] Square: *OkHttp*, https://square.github.io/okhttp/ (2021. dec.)

[12] Square: *Moshi*, https://github.com/square/moshi (2021. dec.)

[13] epec.com: *Battery Cell Comparison*, https://www.epectec.com/batteries/cell-comparison.html (2021. dec.)