Szegedi Tudományegyetem

Informatikai Intézet

DIPLOMAMUNKA

Bányai Balázs

2021

Szegedi Tudományegyetem
Informatikai Intézet

Növény monitorozó rendszer üvegházi alkalmazásokra

Diplomamunka

Készítette:
Bányai Balázs

Mérnökinformatika szakos hallgató

Témavezető:
Dr. Mingesz Róbert Zoltán
egyetemi adjunktus

Szeged
2021

Feladatkiírás

Manapság egyre nagyobb hangsúly kerül a termelési hatékonyság növelésére. Nincs ez másképpen a növények termesztésénél sem. A hatékonyság növelésének egy módja lehet az, hogy a növényeket és környezetüket folyamatosan megfigyeljük és adatokat gyűjtünk be. Az adatok alapvetően jelentős információkat tartalmaznak, de a hatékonyság nagy mértékű növelése érdekében érdemes az adatok változását valamiféle beavatkozás eredményeként tekinteni. A fent leírtak alapján a munka célja egy olyan rendszer kifejlesztése, ami folyamatosan monitorozza a növényeket és környezetüket üvegházi környezetben. Az adatokat adatbázisban tárolja, és a végfelhasználónak értelmezhető módon megjeleníti egy weboldal segítségével. Az adatok mérését és továbbítását mikrovezérlők végzik. A mikrovezérlők az üvegházi környezet miatt vezetéknélküli kommunikációval továbbítják az adatot. A megvalósításhoz felhasználható eszközök: ARM Cortex M és ESP32 mikrovezérlő, adatbázis, webes keretrendszerek.

A munka során a következő alapvető funkciók megvalósítása szükséges:

- Weboldal készítése, ahol a felhasználó áttekintheti a mért adatokat.
- Felhasználói szintek kezelése (csak olvasás, paraméterek írása)
- API készítése az adatok feldolgozására
- Adatok adatbázisban való tárolása
- Növények és környezet monitorozása, adatok továbbítása
- Berendezés védelme a környezeti hatásokkal szemben.

Tartalmi összefoglaló

A téma megnevezése:

Növény monitorozó rendszer üvegházi alkalmazásokra.

A megadott feladat megfogalmazása:

A feladat egy növény monitorozó rendszer megvalósítása. A rendszer több részből áll. Egyik része képes a növény és a környezet különböző fizikai tulajdonságait mérni. A másik része fogadja az adatot, eltárolja, majd meg is jeleníti azt egy végfelhasználó számára egyszerűen kezelhető weboldal segítségével.

A megoldási mód:

A feladat megoldásához szükségem volt egy teszt környezetre. Mivel a növény magasságát vertikálisan mértem, így szükség volt egy állványra, aminek segítségével a mérést meg tudtam valósítani. Ezen felül kellet egy növény, amin a méréseket végre tudtam hajtani. Az jelenlegi időszakra való tekintettel egy szobanövényt választottam.

Alkalmazott eszközök, módszerek:

A rendszer magvalósításához szükségem volt két mikrovezérlőre, továbbá a méréshez nélkülözhetetlen szenzorokra. Az adatok tárolására a MongoDB felhő alapú szolgáltatását választottam. Az adatok feldolgozását és megjelenítését JavaScript keretrendszerek biztosítják.

Elért eredmények:

A Feladat kiírásban szereplő követelményeket sikerült megvalósítanom. A növény monitorozó rendszert teszteltem az általam megalkotott teszt környezetben. A feladat során egy olyan rendszert sikerült megvalósítani, ami valós üvegházi környezetben is megállná a helyét.

Kulcsszavak:

növény monitorozás, mikrovezérlő, VueJS, NodeJS, MongoDB, Szenzorok

Tartalomjegyzék

Bevezetés

Napjainkban egyre nagyobb hangsúlyt fektetünk arra, hogy minél hatékonyabban és minél kevesebb pazarlással végezzük a termelést. Annak érdekében, hogy a hatékonysági fokot növeljük, méréseket kell végezzünk, hogy legyen viszonyítási alapunk. A diplomamunkám kifejezetten a növények üvegházi környezetben való termesztésére összpontosít.

Két nagyobb területen is alkalmazható az általam elkészített monitorozó rendszer. Az egyik ilyen terület a nagy mennyiségben való termesztés, ahol az elsődleges cél a termés mennyiségének maximalizálása. Erre az alkalmazásra hasznosnak bizonyulhat egy rendszer, ahol figyelemmel lehet követni a növény és a környezet különböző tulajdonságait. A másik alkalmazása pedig a genetikailag módosított növények figyelése. A közeljövőben a klíma változás hatására nagy szerepet fog kapni a genetikailag módosított növények előállítása. Az általunk még nem ismert növények minden változását figyelemmel szeretnénk követni annak érdekében, hogy meg tudjuk mondani a genetikai módosítások sikerességét.

1. Adatbázis és szerver oldali technológiák

Az adatok tárolására a választásom a MongoDB-re esett. Az adatbázissal való kommunikációt pedig egy NodeJS keretrendszer segítségével valósítottam meg. Bármilyen tárolt adat lekérdezése vagy módosítása csak ezen a NodeJS alkalmazásprogramozási felületen (későbbiekben: API) keresztül történhet meg a rendszerben, leszámítva a direkt adatbázis műveleteket.

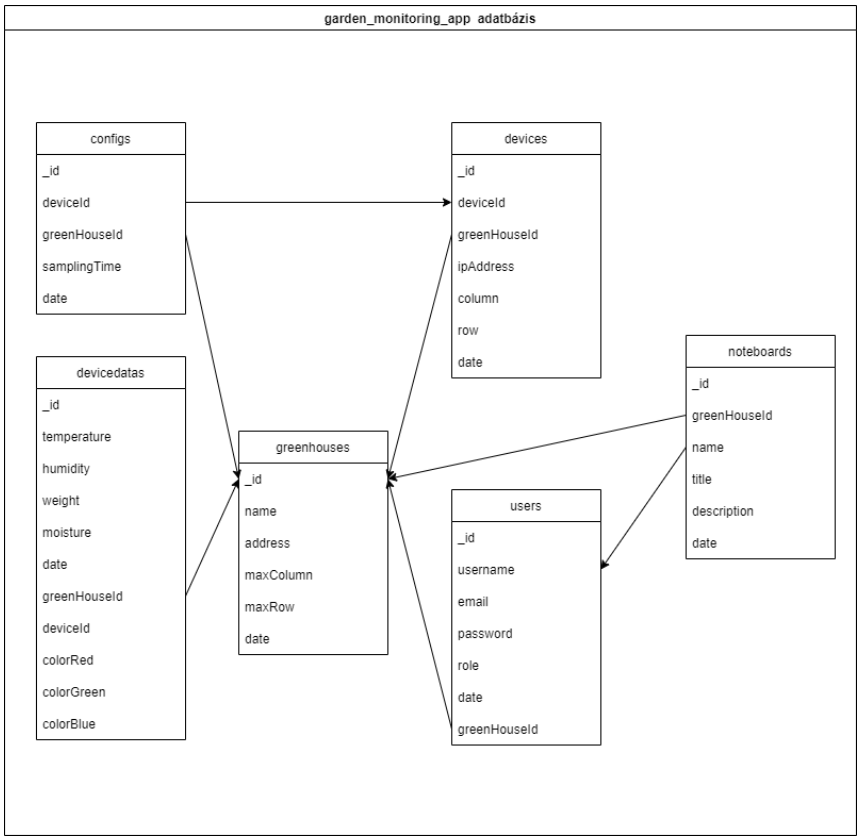
1.1 MongoDB

A MongoDB egy nyílt forráskódú dokumentum adatbázis. A dokumentum adatbázisokat nem relációs, vagy más néven NoSQL adatbázisoknak is szokás nevezni. Ennek előnye az, hogy nem adat táblákban kell tároljuk az adatokat egy előre megadott modell alapján, így sokkal rugalmasabb adat struktúrákat tudunk benne letárolni. Emellett nem szükséges az összes adatnak rendelkeznie ugyan azokkal az adat tagokkal. A MongoDB széleskörűen elterjed a programozók körében, mivel használata nagyon egyszerű, sokkal kevesebb megkötéssel rendelkezik, mint egy relációs adatbázis. Az adatbázis tervezési fázisban töltött idő nagy mértékben rövidülhet. Ez főleg azért lehetséges, mert ahogy azt már említettem, az adatbázis táblák nem egy kötött struktúrát követnek. Továbbá nem szükséges adatbázis normalizálási lépéseket végeznünk. A NoSQL adatbázisok egyik nagy előnye, hogy nagy mennyiségű adat tárolására is képesek drasztikus sebesség csökkenés nélkül. Amiatt, hogy nincs szükség több tábla összefűzésére, a lekérdezéseink is gyorsabban végre hajtódnak, mint egy relációs adatbázisban. Az én rendszeremben nagyon sok mérési adat keletkezhet, ezért döntöttem úgy, hogy ezt az adatbázist fogom használni. [1, 2, 3, 4]

A MongoDB a relációs adatbázisoktól eltérően dokumentumok tárolására képes. A dokumentumok JSON alapúak, ami egy széleskörűen elterjedt adat formátum. A JSON a JavaScript Object Notation angol kifejezés mozaik szava. Szöveges alapú, programozási nyelv független és emberi szemmel is könnyen értelmezhető. A kis, de jól definiált szabályai miatt könnyen kezelhető. A relációs adatbázisoktól eltérően itt nem táblák, hanem úgynevezett collectionok (gyűjtemények) vannak, amelyek tárolják a logikailag összefüggő dokumentumokat. [5]

1.2 MongoDB Atlas

A MongoDB Atlas egy felhő szolgáltatás, amit a MongoDB kínál. Az egyik nagy előnye, hogy nincs szükség egy külön szerver gépet fent tartanunk annak érdekében, hogy a növény monitorozó rendszerünk adatbázisa folyamatosan elérhető legyen. Ahogy az minden felhő szolgáltatásra jellemző, ebben az esetben is egy skálázható szolgáltatást kapunk. Csak annyi erőforrást használ az adatbázis, amennyire az adott pillanatban szüksége van. Ha növeljük az adatok mennyiségét, vagy esetleg az adat forgalmat, akkor a felhő szolgáltató számunkra nem észrevehető módon bővíti a felhasználható erőforrásokat. Az Atlas folyamatos backup mentéseket hajt végre időközönként annak érdekében, hogy az adataink ne vesszenek el valamiféle meghibásodás következtében. Az adatbázis elérése szempontjából is előnyös lehet számunkra a felhő szolgáltatás, mivel a világ teljes területén biztosítanak szervert számunkra. A MongoDB Atlas önmagában nem nyújt felhő szolgáltatást, de a legnagyobb felhő szolgáltató cégekkel együtt működve több szolgáltatón keresztül is lehetséges a szolgáltatás igénybevétele. Igénybe vehetjük az Amazon Web Service (AWS), Microsoft Azure vagy akár a Google Cloud Platform szolgáltatásait is. Sőt, lehetőségünk van akár több szolgáltatást is használni egyszerre. A diplomamunkámban az AWS szolgáltatás választottam, mivel régionként különböző árazással rendelkeznek és esetemben bizonyos korlátokig ez az opció ingyenesen használható. A rendszer bővülésével, mind az adat mennyiség, mind a területi eloszlást figyelembe véve érdemes lehet megvizsgálni, hogy melyik szolgáltatás a legelőnyösebb számunkra. A szolgáltatók sok esetben más mennyiségek szerint készítik az árszabásukat. Valahol az adat mennyisége, valahol a szerverrel való kommunikációk számát mérik, de további mérőszámokat is alkalmaznak. A MongoDB Atlas online kezelőfelületén lehetőségünk van figyelni, hogy az adatbázisunkhoz az elmúlt időben hányan csatlakoztak és mekkora adatforgalmat generáltak. Az adatforgalom kijelzésénél külön választva nézhetjük az írt és olvasott adat mennyiségeket is. [6, 7]



1.2 1. ábra Adatbázis felépítése

1.3 NodeJS

A NodeJS egy JavaScript runtime ami alapvetően aszinkron működésre lett tervezve szerver oldalon. Olyan alkalmazásoknál, ahol szerver oldalon nem szükséges nagy számításokat végezni, előszeretettel használják egyszerűsége és gyorsasága miatt. A fejlesztőknek nem kell foglalkozniuk a szálkezeléssel, mivel a kérések nem blokkolják egymást. A másik nagy előnye, hogy egyszerűen összekapcsolható MongoDB adatbázisokkal. Az egyik legnagyobb hátránya, hogy relációs adatbázisokkal viszont nehezebben összekapcsolható, mivel a kapcsolódáshoz tartozó függvény könyvtárak fejlesztése lassabban halad, mivel a felhasználói igény sem túl nagy ezeknek a fejlesztésére. A NodeJS által használható függvénycsomagok telepítése egy úgynevezett package manager-en keresztül történik, aminek a neve Node Package Manager (npm). A package managerre úgy is tekinthetünk, mint egy nyílt közösségi függvénycsomag tárhely. [8, 9]

1.4 Express JS

Az Express JS egy ingyenes, nyílt forráskódú NodeJS keretrendszer, aminek segítségével gyorsan és egyszerűen tudunk JavaScript alapú szerver oldali alkalmazásokat fejleszteni. A növény monitorozó rendszer esetében ezt a keretrendszert használtam arra, hogy készítsek egy olyan szerver oldali JavaScript API-t, ami HTTP protokollon keresztül hívható, és feladata az adatbázis és a weboldal/mikrovezérlő közti kommunikáció megteremtése. Mivel az adatokat kis mértékben módosítjuk, és vannak olyan végpontok, amik nem csak adatbázisban való módosításra hivatottak így szükségünk volt egy köztes rétegre és ez a köztes réteg az Express API. [10, 11]

1.5 Szerver oldali Express alkalmazás létrehozása

Első lépésként az eszközünkre telepítenünk kell a már előbbiekben említett NPM package manager-t, majd a NodeJS-t. A projekt létrehozását könnyen meg tehetjük Windows operációs rendszeren egy Windows Power Shell vagy egy Command promt segítségével. Linux vagy MacOS operációs rendszernél pedig egy terminál segítségével. Az *npm init* parancs kiadásával létrehozzuk a projektünk alap tulajdonságait leíró *package.json* file-t. A parancs kiadása után meg tudunk adni olyan tulajdonságokat, hogy mi a projekt neve, ki a szerző, mi a készülendő projekt verzió száma, rövid leírása, belépési pontja és még sok más a projektet leíró tulajdonságot.

A *package.json* file a későbbiekben nem csak a projekt alapvető szöveges tulajdonságait fogja leírni, de itt fogja tárolni a projektben használt függőségek listáját is. Ha a projektet meg szeretnénk osztani másokkal, esetleg valamilyen verziókezelőt használunk, akkor fontos, hogy ezt a file-t mindenképpen mellékeljük. Bevett szokás, hogy a *node_modules* könyvtárat, ahol a valós függvénycsomagok tarolódnak nem osztjuk meg, mivel sok kis méretű állományt tartalmaznak, amely elérhető az NPM package manager segítségével, így feleslegesen terhelnénk a tárhelyet. Emellett a függőségek listájában a szükséges csomagok verziószáma is el van tárolva, ezzel biztosítva azt, hogy másik eszközön való üzembe helyezés során ne lépjenek fel verziószám különbség által keletkező problémák. Másik eszközön való beüzemeléshez elegendő belépnünk abba a könyvtárba, ahol a *package.json* file található és parancssorból ki kell adjuk az *npm install* parancsot.

Az Express JS keretrendszer telepítése sem különbözik a többi függvénycsomag telepítésétől. A csomagok telepítése a következő paranccsal történik *npm install <függvénycsomag neve><@verziószám>*. A „@” jel és a verzió szám megadása opcionális. Abban az esetben, ha ezt nem adjuk meg, akkor a legfrisseb elérhető verzió kerül telepítése. Ezen felül van lehetőségünk a csomag telepítésére lokálisan a projekthez és globális a számítógépünkre. Alapvetően a projekthez fogja telepíteni, de a *-g* kapcsoló megadásával az eszközünkre is tudjuk telepíteni. Ez akkor lehet előnyös, ha több projekten is dolgozunk, ami ugyan azokat a csomagokat használják és ugyan azon verziószámmal. Ezek alapján az Express JS telepítése a projektünkhöz az *npm install express* paranccsal történik.

2. A rendszerhez tartozó Express API

Elsőnek létre kell hozzuk az *package.json*-ben megadott belépési pontot, ami esetemben az *index.js* file. Mielőtt elkezdnénk a fejlesztést érdemes hozzá adnunk a projektünkhöz egy csomagot, aminek az a neve, hogy nodemon. Ez a csomag fejlesztés alatt segít majd minket. A nodemon lényege az, hogy nem kell minden változtatásnál újra indítanunk az alkalmazást, mert minden file változás esetén ő ezt automatikusan elvégzi, ami nagyban gyorsítja a fejlesztési folyamatot.

Az express keretrendszer használatához jeleznünk kell, hogy használni fogjuk az express csomagot. A használathoz a require() függvényt kell használnunk, a dolgunk csak annyi, hogy a függvény paraméterében String formában meg kell adjuk a használni kívánt csomag nevét. Az express esetén szükségünk van egy példány létrehozására is, amit a későbbiekben használni fogunk.

```
const express = require('express')
const app = express()
```

2. 1. ábra Express példány létrehozása

Ez után tudjuk használni az *app* példány függvényeit. Ahhoz, hogy elérjük az API-t be kell állítsunk egy portot, amit figyel és itt fogja várni a HTTP kéréseket. Ezt az app.listen() függvény segítségével tehetjük meg. A függvény egy Number típusú paramétert vár, ami az általunk megadott portot fogja jelenteni. Esetemben ez a 3000-es

port lesz, de bármilyen nyitott port megadása lehetséges. Ezután az alkalmazásunkat az *npm start* parancs kiadásával tudjuk futtatni a nodemon miatt, és futása alatt végig figyeli a 3000-es porton érkező kéréseket.

2.1 Mongoose

Miután már van egy futó alkalmazásunk, létre kell hozzuk a kapcsolatot az adatbázisunkkal. A kapcsolat létrehozáshoz és minden adatbázis művelethez a mongoose csomagot használtam, amely kifejezetten a MongoDB és a NodeJS alkalmazások összekötésére lett kitalálva. Használata nagyon egyszerű és a csomag nagyon jól dokumentált.

Használatához hasonló módon a require() függvényt kell használnunk. Az adatbázishoz való csatlakozáshoz a mongoose.connect() függvényt tudjuk használni, amelynek az első paramétere az adatbázis elérési útja. Ezt könnyen meg tudjuk nézni a MongoDB Atlas kezelőfelületén. Lehetőségünk van kiválasztani, hogy milyen programozási nyelven készítjük az alkalmazást, és miután ezt meg tettük, meg kapjuk az elérési utat, sőt még egy rövid példa kódot is a beüzemeléséhez. A mongoose.connect() függvénynek van egy második opcionális paramétere amellyel egyéb adatbázis kapcsolathoz köthető beállításokat lehet megadni. Ezen opciók listája megtalálható a mongoose dokumentációjában. A monitorozó rendszerben nem használok az adatbázis elérési úton kívül semmilyen egyéb beállítást.

A mongoose segítségével létre tudunk hozni adat sémákat, amelyben meg tudjuk adni, hogy a dokumentumok milyen attribútumokkal fognak rendelkezni. Az attribútumokhoz meg tudjuk adni a nevüket, hogy milyen típusúak (csak JavaScript beépített típusok), hogy kötelezően meg kell-e adjuk és ha szükséges, akkor egy alap értéket is. Továbbá a séma második paramétereként egy verzió kulcs is megadható, de én ezt az opciót nem használom. Az így definiált sémából létre tudunk hozni egy úgynevezett mongoose modelt. A mongoose model létrehozásához szükségünk van egy collection névre és az előbbiekben leírt mongoose sémára.

2.2 Jogosultsági körök

Az egész monitorozó rendszerben három féle jogosultsági kört különböztetünk meg. Ezek szükség esetén a későbbiekben könnyen bővíthetőek. Az egyik jogosultsági kör a „User”,

aki egy sima felhasználó. A User képes lekérdezni monitorozási adatot a hozzá rendelt üvegházból, de beavatkozni sehova se tud. A második jogosultsági kör az „Admin”, aki egy üvegház vezetőjének felel meg. Lehetősége van az üvegházhoz tartozó monitorozási adatok lekérdezésére. Ezen felül kezdeményezhet egy azonnali mérést az üvegházhoz rendelt mikrovezérlőn. Továbbá lehetősége van az üvegház adatainak módosításához, vehet fel új eszközöket a saját üvegházhoz, módosíthatja a meglévő eszközök konfigurációját és kezelheti az üvegházhoz rendelt felhasználókat. Az utolsó jogosultsági kör a „SuperAdmin”. Valójában ez egy rendszer üzemeltető jogkör. A monitorozási adatokhoz nem fér hozzá. Szerepe, hogy új üvegházakat vegyen fel és az üvegházakhoz Adminokat rendeljen.

2.3 JWT

A JWT az angol JSON Web Token kifejezésből ered. Szerepe a biztonság növelése, emellett használható a felhasználók azonosítására is. Az express alkalmazásban a jsonwebtoken csomag segítségével tudjuk használni. A bejelentkezési fázisnál az alkalmazás generál egy JWT tokent, amely bizonyos végpontok hívásánál nélkülözhetetlen. Ha ezeket a végpontokat JWT nélkül szeretnénk hívni, akkor hiba üzenetet kapunk válaszul a kérésünkre. A JWT generáláshoz HS256 titkosítási algoritmust használ, ami egy szimmetrikus titkosítás. A kulcs generáláshoz szükségünk van egy úgynevezett secretre. Az alkalmazásomban erre létre hoztam egy külön *config.js* file-t, amiben ezt könnyedén tudjuk változtatni.

```
const token = jwt.sign( payload: { id: user.id }, config.secret, options: {
  expiresIn: 86400 // 24 óra
});
```

2.3 1. ábra JWT token generálása

Az ábrán látszik, hogy a jsonwebtoken csomag sign() függvény segítségével tudunk tokent generálni. Három paramétert vár, az első a felhasználó azonosítója, a második az előbbiekben említett secret és a harmadik paraméterben több beállítás is megadható. Én egyedül a lejárati időt adtam meg, ami jelenleg 24 óra. A token lejáratával újból be kell jelentkeznünk és kapunk egy új tokent, ami további 24 óráig érvényes lesz.

Ahogy azt írtam a biztonságon kívül lehetőségünk van a token által a felhasználók azonosítására is. Az *authJwt.js* file úgy működik, mint egy köztes réteg. Azon végpontoknál, ahol szükséges a JWT token, meg tudjuk adni, hogy milyen jogosultsági szinttel rendelkező felhasználók férnek hozzá az adott végponthoz. A tokent a HTTP kérés header részében küldjük el. Az említett köztes réteg pedig megvizsgálja az érvényességét a jwt.verify() függvény segítségével, aminek két paramétere van, az első a token, a második pedig a generálásánál alkalmazott secret. Ha a generálásnál és az ellenőrzésnél megadott secret nem egyezik meg, akkor nem tudjuk validálni a tokent. Ha valamilyen felhasználó szerepkör is tartozik a végponthoz, akkor annak meglétét is ellenőrzi. Az ellenőrzés úgy történik, hogy a tokenből meg tudjuk állapítani a felhasználói azonosítót. Ha ez sikeresen megtörtént, akkor az adatbázisból lekérdezzük az adott felhasználót, majd, ha ez megtörtént, akkor megvizsgáljuk, hogy milyen jogosultsági szinttel rendelkezik. Ha bármelyik lépésnél hibát észlelünk, akkor nem engedjük tovább a végpont hívást és szöveges hiba üzenetet küldünk válaszként. [13, 14]

2.4 User model és végpontok

A user végpontok a felhasználóval kapcsolatos műveletekért felelősek. A user végpont hívása a következő útvonalon lehetségesek „<szerver címe>/api/user”. A user model felépítése a következő:

- username (String, required)
- email (String)
- password (String, required)
- role (String, default: ’User’)
- greenHouseId (String)
- date (Date, default: Date.now)

A fent említett modelben a required mezőket kötelező megadnunk, míg a default kulcsszóval rendelkező mezőknél, ha nem adunk meg nekik értéket, akkor ezt alapértelmezetten meg kapják az adatbázis dokumentum létrehozásánál. A Date.now egy JavaScript függvény, aminek a segítségével az adatbázisba való beszúrás időpontját tárolhatjuk.

A „/api/user/” végpont HTTP GET metódus hívására csak bejelentkezett felhasználók jogosultak. Ezen felül vagy Admin vagy SuperAdmin jogosultsági körrel

rendelkező felhasználóknak engedélyezett. Ha a lekérdezést egy Admin jogosultsági körrel rendelkező felhasználó hívja meg, akkor azokat a felhasználókat kapja válaszul, akik ugyan azzal a greenHouseId-val rendelkeznek. Abban az esetben, ha egy SuperAdmin hívja meg, akkor az összes felhasználót meg kapja válaszul.

A „/api/user/userByToken” végpont HTTP GET metódus hívására csak bejelentkezett felhasználók jogosultak. Ez a végpont azt a célt szolgálja, hogy lekérdezzük a weboldalon már bejelentkezett felhasználó felhasználónevét, email címét és a jogosultságát. Abban az esetben, ha nem található felhasználó az adatbázisban a kérés headerjében szereplő token alapján, hibát küldünk vissza a válaszban.

A „/api/user/login” végpont HTTP POST metódus hívás a felhasználó bejelentkezéséért felel. Mivel a bejelentkezésnél még nincs tokenünk, így nincs szükség tokenre a bejelentkezéshez. A HTTP kérés törzsében várja a username és a password adattagokat. Ha a username nem található az adatbázisban, akkor hiba üzenetet küldünk, hogy a felhasználó nem létezik. Ha a felhasználót megtaláltuk, de nincs hozzá rendelve üvegház, akkor szintén hiba üzenetet küldünk, mivel csak olyan felhasználókat szeretnénk beengedni a rendszerbe, akik ezzel rendelkeznek, kivéve, ha a jogosultsági köre SuperAdmin. Ha eddig eljutott az adatok feldolgozása, akkor megvizsgáljuk, hogy a jelszó meg egyezik-e az adatbázisban lévővel. Az adatbázisban a jelszót titkosítva tároljuk. Abban az esetben, ha minden sikeres volt legeneráljuk a JWT tokent és válaszként visszaküldjük a felhasználó összes adatát és a tokent.

A „/api/user/register” végpont HTTP POST metódus hívására történik meg a felhasználó regisztrálása. Mivel a regisztrációs adatoknál nem csak egyezőséget kell vizsgáljunk, ezért használok egy express-validator NPM csomagot. A validálásnál olyan kritériumokat szabtam meg, hogy a jelszó hossza legalább öt karakter kell legyen, az emailnek meg kell feleljen az email szabványokan és a felhasználónév nem lehet üres. Ezen felül egyik sem tartalmazhat speciális karaktereket és a felesleges whitespaceket az elején és végén is levágom. Minden egyes kritériumra hiba üzenetet küldünk tömb formájában, hogy ne csak az első hibát lássa a felhasználó abban az esetben, ha több kritériumban is hibás adatot adott meg. A server oldali validálás nagyon fontos, mivel az informatikában jártasabb emberek nem csak az alkalmazáshoz készített weboldalon keresztül tudnak API végpont hívásokat intézni. Így abban az esetben is csak helyes adatokra végzünk adatbázis műveletet, ha valaki direkt végpont hívást alkalmaz. Ha sikerült helyes adatokat megadni a felhasználónak, akkor a jelszavát az adatbázisba írás előtt titkosítjuk. A titkosítás a bcrypt NPM csomag segítségével történik. Egy több körös

titkosítás használ, esetemben a körök számát egy *saltRounds* nevű konstansban tárolom, aminek az értéke 10.

A módosításhoz és törléshez HTTP PUT és DELETE metódusokat használom a „/api/user” végponton. Mindkét végpont hívás esetén Admin vagy SuperAdmin jogosultságra vagy szükségünk. Emellett Admin esetében csak azon felhasználókra hívhatja meg ezeket a végpontokat, akik ugyan ahhoz az üvegházhoz vannak rendelve. A törléshez a HTTP üzenet törzsében csak egy *id* attribútumok kell elküldenünk. Abban az esetben, ha ezt meg találja az adatbázisban, akkor azt ki törli. A módosítás egy kicsit másabb. Itt is kötelező az *id* megadása, de mellette el kell küldjük azokat az attribútumokat is, amiket változtatni szeretnénk.

2.5 GreenHouse modell és végpontok

A greenHouse végpontok az üvegház műveletekért felelnek. A végpontok hívása a következő útvonalon lehetséges „<szerver címe>/api/greenHouses”. A GreenHouse model a következőképpen néz ki:

- name (String, required)
- address (String)
- maxColumn (Number, required)
- maxRow (Number, required)
- date (Date, default: Date.now)

Az maxColumn és a maxRow az üvegház mátrix szerű felosztására szolgál. A későbbiekben az üvegházak adott oszlopaihoz és soraihoz eszközöket fogunk rendelni. A felosztás segíti a tájékozódást és abban az esetben, ha nem rendelkezünk elég eszközzel, akkor lehetőségünk van egy redukált monitorozásra, ahol egy mátrix cellát tekinthetünk egységesnek. Megfelelő méretű cellák esetén nagyon hasonló körülményeknek vannak kitéve a növényeink, ezért valószínűleg hasonló eredményt kapnánk egy cellán belül lévő eszközök esetén. Ez lehet egy költség csökkentési szempont is, mivel a pontosság csak kis mértékben romlana, de a költségek jelentős mértékben csökkennének. Ezt a felvetés a későbbiekben érdemes lehet tesztelni éles nagyipari környezetben.

Az üvegházak listázásához, létrehozásához, törléséhez és módosításához a „/api/greenHouses” végpontok hívhatok HTPP GET/POST/PUT/DELETE metódusok használatával. Mindegyik metódus esetén szükséges a bejelentkezés és a SuperAdmin

jogosultsági szint. A GET hívás esetén minden létező üvegházat megkapunk válaszul. A POST hívás esetén a modelben szereplő összes kötelező attribútumot tartalmaznia kell a kérés törzsének, különben hibát kapunk. A PUT hívás esetén az üvegház *_id* attribútumnak kell szerepeljen, ami az adatbázis által létrehozott dokumentum azonosítója. Ezen felül el kell küldjük az összes módosítani kívánt adat tagot. A DELETE hívás esetén csak egy *id*-t kell küldjünk.

Lehetőségünk van egy adott üvegházban elérhető maximum oszlop és sorszám lekérdezésére is, amit a weboldal fog használni arra a célra, hogy kiválasszunk egy üvegház cellát. Ezt a „/api/greenHouses/maxColAndRow” végponton érhetünk el. Az eléréséhez nem szükséges jogosultság, csak bejelentkezés. Nem vár semmilyen adatot, mivel a felhasználó tokenen keresztül tudja az adott üvegházat. A lekérdezésünk válaszul a *maxColumn, maxRow* és a *deviceNumber* adatot küldi vissza. A *deviceNumber* az üvegházhoz rendelt összes eszköz száma.

A greenHouse végpontok között szerepel egy olyan végpont, ami az üvegházakat kérdezi le, de csak egy redukált formában. Visszatérése egy tömb, amiben csak az *id* és a *name* van. Ez biztonsági és adatforgalmi szereppel is jelentkezik. Minden végpont esetén törekedtem arra, hogy csak azokat az adatokat adják vissza, ami az adott funkcióhoz kell. Ezt a „/api/greenHouses/getGreenHouseListForVue” végponton lehet elérni SuperAdmin jogosultsági szinttel.

2.6 NoteBoard model és végpontok

A weboldalon megtalálható egy üzenőfal, melynek az a szerepe, hogy a növények gondozásában résztvevő emberek információkat oszthassanak meg egymással. Fontos lehet, hogy legyen egy közös felület, amely csak egy adott üvegházhoz tartozik így fontos információkat tudnak egymással megosztani a gondozók. A NoteBoard model a következőkből áll:

- name (String, required)
- greenHouseId (String)
- title (String, required)
- description (String, required)
- date (Date, default: Date.now)

A name mező tárolja a felhasználó nevét, a title a bejegyzés címét és a description a bejegyzés lényegi részét.

A bejegyzéseket nem lehet módosítani vagy törölni, hogy az üzenet folyam folyamatos legyen, és az üvegház felelőse részletesen lássa az eseményeket. A „/api/noteBoard” GET metódus hívásra dátum szerint csökkenő sorrendben meg kapjuk az összes bejegyzést. A végpont eléréséhez a felhasználó bejelentkezése szükséges, mivel a token segítségével állapítjuk meg a felhasználóhoz tartozó üvegházat és csak a felhasználóhoz tartozó üvegházhoz rendelt bejegyzéseket szeretnénk megjeleníteni. Ugyan erre a végpontra hívott POST metódus segítségével új bejegyzést hozhatunk létre. A kérés törzsében szerepelnie kell a *title* és a *description* tagoknak.

2.7 Config model és végpontok

A Config végpontnak az a szerep, hogy az eszközök bizonyos beállításait futás közben dinamikusan tudjuk állítani. Jelenleg ennek a mintavételezési időben van szerepe. Meg tudjuk adni, hogy az eszközök egyenként milyen sűrűn küldjenek mérési adatokat. A config model a következő:

- deviceId (Number, required)
- greenHouseId (String, required)
- samplingTime (Number, default: 10)
- date (Date, default: Date.now)

A samplingTime a mintavételetzési időközt jelenti percben megadva.

A mikrovezérlőhöz tartozó konfiguráció lekérdezése a „/api/config” végpont GET lekérdezéssel lehetséges. Lekérdezési paraméterben meg kell adjuk a *deviceId*-t.

A weboldal segítségével ezt módosíthatjuk is. Ezt az előző végpont POST hívásával tehetjük meg. A hívásban ugyan azokat az adatokat kell megadjuk, csak a lekérdezés törzsében. Ha sikerült megtalálnunk ezt a konfigurációs ad azatbázisban, akkor lekérdezzük a hozzá tartozó eszköz IP címét is, amit szintén tárolunk az adatbázisban. Annak érdekében, hogy az eszköz frissítse a mintavételezési időközt, valahogy tájékoztatnunk kell a változásról. Hogy ezt meg tudjam valósítani az eszközön fut egy HTTP szerver, ami folyamatosan várja a hívásokat. Ha az eszköz „/config” útvonalára intézünk GET lekérést, akkor ezt fogadja és egy 200-as státusz kódú válasszal tér vissza. Ezzel jeleztük a változást és a többi teendőt az eszköz végzi.

Mivel az eszközöknél nem használunk tokeneket, így nem tudjuk a tokenből meghatározni a *greenHouseId*-t. Egy külön „/api/config/deviceGetData” végpontot kellett létrehozni annak érdekében, hogy a mikrovezérlő is le tudja kérdezni a hozzá tartozó konfigurációt. Ez annyival másabb, hogy itt lekérdezési paraméterben meg kell adjuk a *greenHouseId*-t is a *deviceId*-n kívül.

2.8 Device model és végpontok, DeviceData model.

A Device végpontok egyszerre szolgálnak az eszközök nyilvántartására és az általuk küldött mérési adatok kezelésére. A Device model a következő:

- deviceId (Number, required)
- greenHouseId (String, required)
- ipAddress (String, required)
- column (Number, required)
- row (Number, required)
- date (Date, default: Date.now)

Az *ipAddress* a konfiguráció változásánál értesítendő út miatt szükséges. A *column* és a *row* az eszköz celláját azonosítja az üvegházban. A DeviceData model a következő:

- deviceId (Number)
- greenHouseId (String)
- laserDistance (Number)
- sonicDistance (Number)
- temperature (Number)
- humidity (Number)
- weight (Number)
- moisture (Number)
- colorRed (Number)
- colorGreen (Number)
- colorBlue (Number)
- date (Date, default: Date.now)

Ezek az adatok arra szolgálnak, hogy az eszközök által érzékelt mennyiségeket el tároljuk, hogy lássuk mi történik a növénnyel és környezetével az idő teltével.

Lehetőségünk van lekérdezni a felhasználó üvegházához tartozó eszközöket a „/api/device” végponton keresztül GET lekérdezéssel. Mivel az üvegház lekérdezése tokenen keresztül történik, így szükségünk van a bejelentkezése, de további jogosultság nem szükséges. Ugyan erre a végpontra küldött POST kérés esetén lehetőségünk van eszköz létrehozására és módosítására, de el kell küldjük a model kötelező mezőit a kérés törzsében. Ez a végpont annyival másabb a többi POST hívástól, hogy sikeres eszköz mentés esetén létre hozunk egy Config dokumentumot is az adat táblában az alap *samplingTime* értékkel. A DELETE metódus hívásra az eszköz törölhető a kérés törzsében megadott *id* alapján.

A „/api/device/deviceList” GET kérés esetén visszakapjuk az eszközök egy redukált listáját a megadott cellában (oszlop és sor páros). A válasz tartalmazza a felhasználó üvegházához tartozó összes eszköz *deviceId*-ját és az adatbázis által létrehozott dokumentum *_id*-t, amik az adott oszlopban és sorban vannak. Az oszlop és sor adatok a lekérdezés paraméterében kell megadjuk.

Mivel az adatokat erősen kötődnek az eszközökhöz, ezért egy végpont csoportban kezeltem az eszközöket és a hozzájuk tartozó adatokat. Az adatok lekérdezésére a „/api/device/deviceData” végpont lett létrehozva. Mindössze annyi dolgunk van, hogy a lekérdezés paraméterének át adjuk az eszköz dokumentum *_id*-jét. Abban az esetben, ha megadjuk paraméterként a *date* tagot is, akkor az adott napra vonatkozó mérési adatokat kapjuk vissza. Ennek a weboldalon lesz nagy szerepe, mivel nem minden esetben szeretnénk az összes mérési adatot látni. Ez akkor lesz előnyös, ha már huzamosabb ideje mér az eszközünk, de mi csak bizonyos napok adataira vagyunk kíváncsiak. Ugyan ebből a megfontolásból, az összes mérési adatot dátum szerint csökkenő sorrendben kapjuk meg.

A weboldalon lesz lehetőségünk arra, hogy azonnali mérést kezdeményezzünk egy eszközzel. Ehhez valahogyan jeleznünk kell a mikrovezérlőnek. Ismét azt a módszert választottam, hogy az eszközön lévő HTTP szervernek küldünk egy üzenetet, hogy kezdje meg a mérést. Ezt a „/api/device/startMeasurment” végpont hívásával tehetjük meg. Paraméterként a *deviceId*-t várja. Meghívása után elküldi az eszköz IP címére a kérést.

A végpontok nagy része eddig a weboldallal volt kapcsolatba hozható. A „/api/device/measuredData” végpont viszont az eszközök számára van fenntartva. Az eszköz elküldi az összes DeviceData model által leírt adatot és ezt módosítás nélkül elmentjük az adatbázisunkba.

3. Monitorozó rendszer kezelő weboldal

Az általam megalkotott weboldal Vue.Js keretrendszer segítségével íródott. Manapság az egyik legelterjedtebb JavaScript alapú keretrendszer. Tökéletesen alkalmas egyszerű weboldalak létrehozására, de lehetőségünk van komplexebb igényeket is megvalósítani segítségével. Mivel komponens alapú, így lehetőségünk vagy egy már meglévő komponens használatára több helyen is. Ez hasznos lehet olyan esetekben, mikor nagyon hasonló elemeket szeretnénk létrehozni. Ennek segítségével elkerülhető a kód duplikáció. Sokkal igényesebb és átláthatóbb kódot adhatunk ki a kezeink közül, ha kihasználjuk a keretrendszer nyújtotta lehetőségeket. A kinézet kialakításában segítségemre volt a bootstrap-vue CSS könyvtár. Segítségével egyszerűen egységes kinézetre hozhatjuk az egész weboldalt. [15, 16]

A vue kiterjesztésű fileok két fő részből tevődnek össze. Az egyik a template rész, ami többnyire HTML kódot tartalmaznak. Lehetőségünk van „közbe ékelni” JavaScript elemeket is. Amikor a felhasználó lekérdezi az adott oldalt, akkor a template részben lévő JavaScript kód HTML kódra fordul és a felhasználóhoz már csak egy kész HTML kódot juttatunk el. A másik fő része a script rész. Itt tudunk létrehozni és módosítani adat tagokat, amit a későbbiekben majd a template részben használni tudunk. Továbbá lehetőségünk van események bekövetkezésére eljárásokat végre hajtani. Ez hasznos olyan esetekben, amikor például kattintásra eseményre szeretnénk műveletet végrehajtani, de nem szeretnénk gombokat használni ennek megvalósításra.

3.1 Belépési pont

A weboldal belépési pontja az *src* mappában található *main.js*. Az állomány elején lehetőségünk van jelezni, hogy milyen csomagokat szeretnénk használni az alkalmazásunkban, ezt az *import* kulcsszóval tehetjük meg. A projekt létrehozásakor már legenerált importálásokon felül saját magam is importáltam pár hasznos csomagot. Az egyik ilyen csomag a már említett bootstrap-vue, amely a weboldalon megjelenő elemek kinézetéért és a reszponzivitásért felel. A következő ilyen csomag a vue-select, amely a lenyíló bemeneti mezők kezelését könnyíti meg. Ezen felül nagyon hasznos csomag az axios. Abban játszik szerepet, hogy könnyedén meg tudjuk hívni az Express API végpontokat és az általa visszaadott adatot. Legvégül egy vuejs-noty csomagot

használtam, amivel letisztult értesítéseket küldhetünk a felhasználók számára. Jelenleg az API végpont hívások sikerességét és sikertelenségét jelzi.

A csomagok importálása után végre kell hajtsunk pár beállítást annak érdekében, hogy egyszerűen tudjuk kezelni az API végpontokat. Be kell állítanunk az API elérési útját, amihez majd a későbbiekben hozzá fűzünk további elérési utakat.

Az alkalmazásban használok LocalStorage-t, ami adatok tárolását teszi lehetővé az oldalt látogató felhasználók böngészőjében. Azért lesz erre szükségünk, mert a felhasználó azonosítására használt tokent le tudjuk itt tárolni és nem szükséges újra bejelentkeznünk az oldalra, ha bezártuk az adott böngésző oldalt. Abban az esetben, ha belépünk a *main.js* belépési ponton keresztül és létezik a LocalStorage-ban token, akkor ezt hozzá adjuk az axios által intézett HTTP lekérdezések fejléc mezőjébe, amit majd az API képes lesz értelmezni és így azonosítani a felhasználót. Ha végeztünk ezekkel a lépésekkel, akkor betöltjük az App.vue állományt.

3.2 Vue store

A vue store lehetőséget ad számunkra, hogy szerver oldalon tároljunk adatokat, amely a teljes alkalmazásban elérhető. Minden egyes felhasználó által megnyitott böngésző oldalhoz létrehoz egy külön store-t, amelyben az adatok csak a megnyitott lapon lesznek elérhetőek. A storeban lévő adatokat módosíthatjuk és lekérdezhetjük. A weboldalamon egyetlen store-t használok, ahol a felhasználó adatait tárolom. Ez a *store* mappában érhető el a *user.js* fileban. Az itt található adatok a következőek:

- username (felhasználónév)
- email (felhasználó email címe)
- token (felhasználó azonosítására szolgáló token)
- role (felhasználó jogosultsági szintje)
- greenHouseId (felhasználóhoz rendelt üvegház azonosítója)
- isLoggedIn (a felhasználó be van-e jelentkezve)

Két nagyobb művelet található meg az adatok mellett. Az egyik az *initUser* amelynek a szerepe, hogy bejelentkezéskor, vagy az oldal megnyitásakor betöltse az adatokat a storeba a token segítségével. A függvény meghívásakor meghívjuk a „/api/user/userByToken” végpontot, ami válaszul pontosan ezeket az adatokat fogja visszaadni és betölti a storeba. Ha a művelet sikertelen, akkor egy hiba értesítést kapunk.

A másik nagyobb művelet a *logout*. Amikor a felhasználó kijelentkezik, akkor szeretnénk, ha minden store-ban lévő adata törlődne. A művelet az összes fentebb említett adatot null értékre állítja.

3.3 App.vue és a router

Az App.vue renderelését a *main.js* végzi el, ami az alkalmazás belépési pontja. Az App.vue létrehozásakor a fentebb említett *initUser* store művelet fut le, aminek az a szerepe, hogy ha a felhasználó már bejelentkezett és a tokene érvényes, akkor töltsük ezt be a storeba és ne keljen újra bejelentkeztetni.

Az App.vue-ban megtalálható egy router-view componens ami azért felel, hogy különböző URL-re különböző oldalakat töltsünk be. Azt, hogy milyen URL-re melyik oldalt kell betölteni, azt a *router* mappában található *index.js* fogja eldönteni.

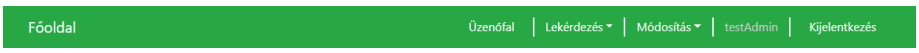
A *routes* konstants tárolja azokat az utakat, ahova el tudjuk navigálni a felhasználót. A *routes* egy tömb, ami objektumokat tárol. Minden objektum egy útvonalért felel. Az objektum *path* tagjában tudjuk meghatározni, hogy milyen útra lépjen életbe. A *name* tag szolgál arra, hogy a projektünkön belül tudjuk navigálni a felhasználót. Ezt meg tehetnénk úgy is, hogy egy sima anchor elemmel át irányítjuk egy másik URL-re, de ebben az esetben az oldal újra töltődik. Ezzel elveszítené az oldalunk a Single Page Application szerepét, amit mi nem szeretnénk. Az átirányításoknál a *name* segítségével úgy tudunk át navigálni egy másik oldalra, hogy nem kell újra töltenünk az oldalt. A *component* tag segítségével tudjuk megadni azt, hogy milyen componenst töltsön be az oldalunk egy adott útvonalra. A *meta* részben az úthoz tartozó speciális tulajdonságokat írtam le. Az itt szereplő tagok fogják meghatározni, hogy egy felhasználó beléphet-e az adott oldalra, vagy sem.

Az előbbiekben említett *meta* taget meg kell vizsgáljuk minden egyes útvonalra való belépéskor. Ezt a *router.beforeEach* függvénnyel tehetjük meg. Ez a függvény minden egyes router esemény előtt meghívódik és meg kapjuk az is, hogy milyen útvonalról milyen útvonalra szeretnénk lépni. A weboldalt egyik szolgáltatását sem lehet igénybe venni mindaddig, amíg nem jelentkezünk be. Kivétel ez alól a bejelentkezés és a regisztráció. Minden más esetben, ha bejelentkezés nélkül szeretnénk meglátogatni egy oldalt, akkor át irányítjuk a bejelentkezés oldalra, hogy kötelezzük a belépésre. Továbbá ebben a függvényben vizsgáljuk meg azt is, hogy a felhasználó jogosult-e az adott oldal látogatására, ha nem, akkor szintén a bejelentkezési oldalra irányítjuk.

Az App.vue *router-view* komponense előtt minden esetben betöltjük a *Header* komponenst.

3.4 Header komponens

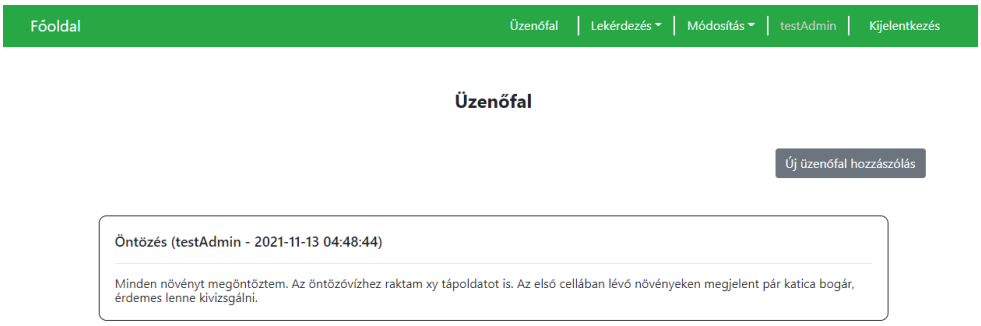
A Header komponens a felső navigációs menüt jeleníti meg. Itt lekérdezzük a felhasználó jogosultsági szintjét a storeból, hogy csak olyan elemeket jelenítsünk meg a felhasználó számára, amihez jogosultsága is van. Három fő menüvel rendelkezik. Az első az üzenőfal. Ez csak a User és az Admin jogosultsággal rendelkező felhasználók számára jelenik meg, mivel SuperAdmin számára ez nem értelmezett. A második menüpont a lekérdezés, ami szintén ezekkel a jogosultsági körökkel elérhető. A harmadik menüpont a módosítás. Ez a menüpont három almenüre bontható. Ez csak az Admin és SuperAdmin jogosultsági körökkel elérhető, mivel a Usernek nincsen jóga módosítani. A menüpontok mellett itt található meg a kijelentkezés gomb is. Megnyomására egy metódus hívás következik be, amelynek hatására kitöröljük a storeban tárolt felhasználói adatokat.



3.4 1. ábra Header komponens Admin jogosultsággal

3.5 Üzenőfal komponens

Az üzenőfal komponensem azt a célt szolgálja, hogy az azonos üvegházhoz tartozó felhasználók üzeneteket osszanak meg egymással. Az oldal létrehozásakor az axios segítségével lekérdezzük az összes bejegyzést, majd megjelenítjük azokat. Az üzenetnél megjelenítjük az üzenet írójának felhasználónevét, az üzenet létrehozásának időpontját és magát az üzenetet. Ezen a felületen van lehetőségünk az üzenet létrehozására is. Az „Új üzenőfal hozzászólás” gomb megnyomására megjelenítünk egy modalt, ahol megadható a fejléc és a szöveg. Ha kitöltöttük a mezőket, akkor a mentés gomb megnyomásával elküldjük és mentésre kerül az adatbázisban. A küldés finally() függvény callback részében újra töltjük az oldalt. Ezzel frissítjük a megjelenítendő üzenetek listáját.

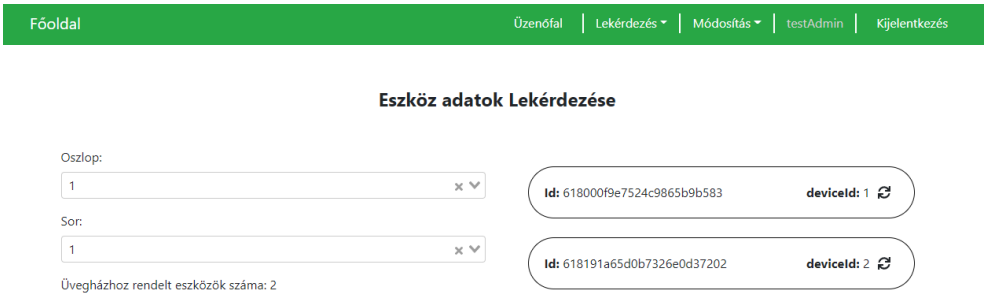


3.5 1. ábra Üzenőfal komponens

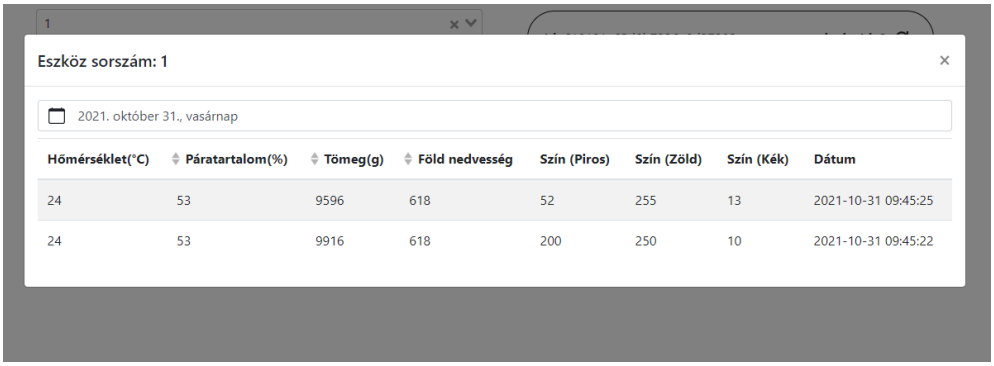
3.6 Lekérdezés komponens

A lekérdezések komponensnél lehetőségünk van megtekinteni a mérési adatokat. Annak érdekében, hogy felhasználói szempontból jól kezelhető legyen, nem csak simán ki listázzuk az összes mérést, hanem ki kell választuk, hogy melyik cellában lévő eszközök adatait szeretnénk látni. A komponenst betöltésénél lekérdezzük, hogy a felhasználóhoz rendelt üvegházban mennyi a maximum oszlopok és sorok száma. Ha ez megtörtént akkor mind a két adatra legördülő listát jelenítünk meg a lehetséges értékekkel. Ezt a két értéket kötelező megadni, ha ezeket megadtuk, akkor megjelennek a cellához tartozó eszközök. Abban az esetben, ha nincs a cellához eszköz rendelve, üzenettel tájékoztatjuk a felhasználót, hogy ebben a cellában nincs eszköz. Az eszközök megjelenítésénél az eszköz dokumentum azonosítója és az eszköz azonosítója szerepel. Admin felhasználóként megjelenik egy ikon is számunkra, amivel azonnali mérést kezdeményezhetünk az eszközön. Ha azonnali mérést kezdeményezünk, akkor 2 másodpercig nem tudjuk újra megnyomni a gombot.

A mérési adatokat úgy tudjuk megtekinteni, hogy rá kattintunk az általunk választott mikrovezérlő adatait tartalmazó területre. Ekkor megjelenik egy modal. A modal megnyitásával elindítunk egy lekérdezést az adott eszköz adataira. Alaphelyzetben minden mérési adatot megjelenítünk az adott eszközhöz. Lehetőségünk van az adatok szűrésére, hogy csak egy bizonyos napon mért adatokat írjon ki. Ez azért kellett, mert ha huzamosabb ideig mérünk az eszközzel akkor rengeteg adat közül kell ki keresnünk a számunkra releváns adatot. A dátum kiválasztására egy dátum választót használok, amelyből törölni is lehet a dátumot, ami azt eredményezi, hogy újra az összes mérési adat megjelenik.



3.6 1. ábra Lekérdezések Admin felhasználóval



3.6 2. ábra Eszköz adatok modal

3.7 Módosítások menüpont

A módosítás menüpont alatt elérhető Admin felhasználók számára a felhasználók módosítása, eszközök módosítása és az eszköz konfigurációt módosítása. SuperAdmin felhasználóként pedig a felhasználók módosítása és az üvegházak módosítása.

A felhasználók módosításánál a már meglévő felhasználókat módosíthatjuk vagy törölhetjük. A módosításnál Admin felhasználóként egyedül a jogosultságot lehet változtatni, míg SuperAdmin felhasználóként üvegházat is tudunk rendelni egy felhasználóhoz. Ennek az az oka, hogy a SuperAdmin hivatott arra, hogy üvegházhoz felvegyen felhasználókat, mivel ő kezeli a rendszert. Felhasználók felvételére azért nincs lehetőség, mert így minden regisztráló a saját titkos jelszavát tudja használni, amit a SuperAdmin nem tud.

Eszközök módosításánál lehetőségünk van eszközöket törölni és új eszközöket létrehozni. Az eszköz módosítása nem értelmezett mivel azzal valótlan képet adnának a mérések. Gondoljunk csak bele abba, hogy egy eszköz egy ideig mér egy adott cellát, majd valaki ezt át állítja. A mérési adatoknál nem vennénk észre azt, hogy az eszköz át

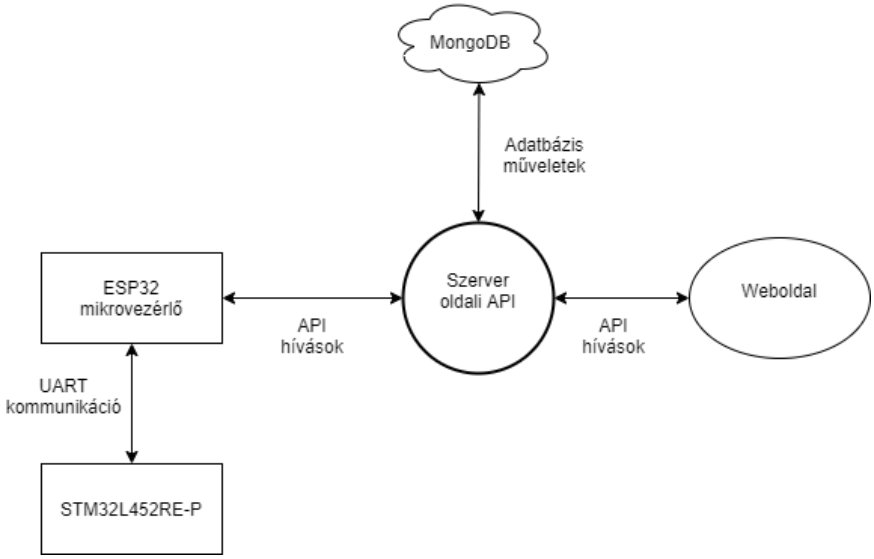
lett helyezve és a mérésekben előfordulhat egy nagy törés a környezeti viszonyok változása miatt.

Az eszközök konfigurációjának módosítása hasonló az adatok lekérdezéséhez. Ki listázzuk az elérhető eszközöket, majd az eszköz adataira kattintva előjön egy modal, ahol előre meghatározott értékek közül választhatunk.

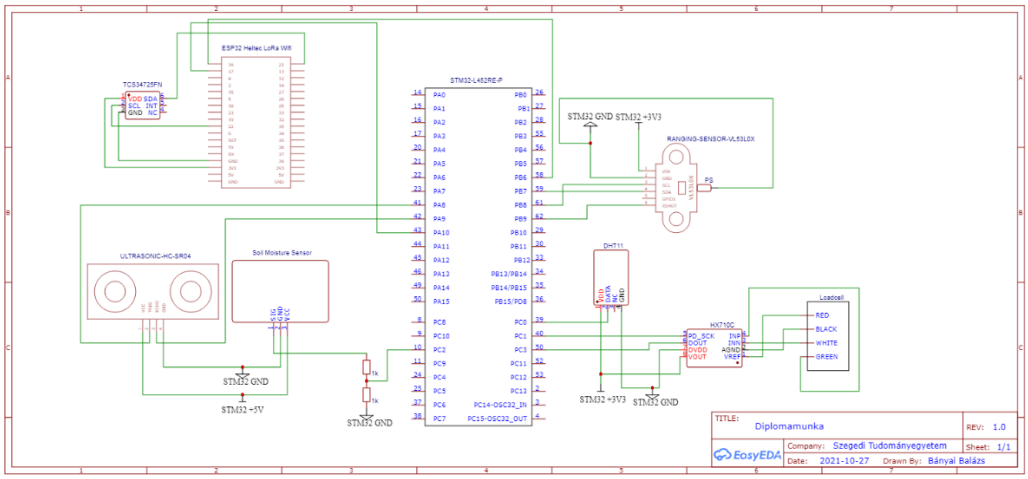
SuperAdmin felhasználóként elérhető számunka az üvegházak módosítása. Itt létre tudunk hozni új üvegházakat, láthatjuk a meglévő üvegházakat, módosíthatjuk azokat és még törölni is tudunk. Itt tudjuk megadni az üvegház cellákra osztásához szükséges oszlopok és sorok számát.

4. Monitorozó eszközök

A növények monitorozásához két mikrovezérlőt használok. Az egyik ilyen mikrovezérlő az STM32L452RE-P a másik pedig a Heltec WiFi LoRa 32 (V1). A mikrovezérlők tökéletes eszközök az általam definiált feladat ellátására. Előnyük, hogy kis helyen elférnek és nagyon alacsony az energia fogyasztásuk. Nagyon sokoldalú eszközök és egyszerűen bővíthetők. A mérnöki feladatok ellátásához sokszor figyelembe kell vennünk a rendelkezésre álló eszközöket. Azért használtam két mikrovezérlőt, mivel az STM32 mikrovezérlő nem tartalmazott WiFi modult, de a már meglévő ESP32 alapú mikrovezérlő igen. A másik előnye, hogy nagyobb jártasságot szereztem a mikrovezérlők közötti kommunikáció megvalósításában is. Abban az esetben, ha a rendszerünket bővíteni szeretnék, figyelembe kell vegyük azt is, hogy mekkora a fogyasztása bizonyos szenzoroknak. Ebben is segítséget nyújthat a két mikrovezérlő használata.



4. 1. ábra Rendszer felépítése



4. 2. ábra Mikrovezérlők kapcsolási rajza

4.1 ARM Cortex M alapú mikrovezérlő

A mikrovezérlő választásánál figyelembe vettem azt, hogy milyen eszközök állnak rendelkezésemre. Ezért esett a választás az STM32L452RE-P típusú mikrovezérlőre.

A mikrovezérlő konfigurálására a gyártó által kiadott STM32CubeMX szoftvert használtam. Lehetőségünk van kiválasztani az általunk használt mikrovezérlőt. Ez alapján csak azok a beállítások lesznek számunkra elérhetők amire az eszközünk képes. A konfigurációk megadása után generálhatunk kódot. A generálásnál ki kell választanunk a fejlesztő környezetet, amiben dolgozni szeretnénk. Én az Atollic Studio-t választottam erre a célra. Miután a generálás megtörtént kód szinten is látjuk a beállításokat, amit az

STM32CubeMX-ben megadtunk. Az automata kód generálás és a fejlesztő környezet előnye az is, hogy a projektünkhöz kezdésből rendelkezésre áll a HAL függvény könyvtár. A HAL függvény könyvtár használatával egyszerűbben kezelhetjük a mikrovezérlőt, mivel nem kell fejlesztés közben memóriacímeket írnunk és olvasnunk. Ez azért lehetséges, mert ezek vannak leimplementálva a függvény könyvtárakban. Az általam írt program C nyelven íródott, de van lehetőség más nyelveken is programozni az eszközt.

4.2 ESP32 mikrovezérlő

Az ESP32 mikrovezérlő széles körben elterjedt eszköz. Olcsón beszerezhető és az árához képest nagyon széleskörűen felhasználható. Mivel a rendszerben vezetéknélküli kommunikációt valósítok meg, ezért egy olyan típusú mikrovezérlőt választottam, amiben beépített WiFi modul található. Az előző mikrovezérlőhöz hasonlóan ez az eszköz is több nyelven programozható. Mivel a rajta futó program bonyolultsága nem nagy, ezért Arduino keretrendszer segítségével programoztam, amit sokan használnak egyszerűsége miatt.

4.3 Távolság érzékelése ultrahang alapú szenzorral

A távolság érzékelését egy HC-SR04 szenzor segítségével valósítottam meg. Az érzékelés azon az elven alapul, hogy az általa kiadott ultrahang vissza verődik a mérendő felületen és érzékeli azt. A szenzor 4 lábbal rendelkezik. A GND a földelés, a VCC a tápfeszültés, ami a szenzor esetében 5V. A mérés megkezdéséhez a Trig elnevezésű lábra kell kiadnunk egy legalább $10\mu s$ hosszúságú impulzust. Ennek hatására nyolc ultrahang jelet ad ki a szenzor. A kiadott jel érzékelése az Echo láb segítségével történik. Az Echo jel a kiadott 8 impulzus után magas értéket vesz fel és mindaddig tartja ezt a magas értéket, amíg vissza nem érkezett az általa kiadott jel. A szenzor mérési tartománya $2cm$ és $4m$ között van. Ez azt jelenti, hogy az adatlap szerint a gyártó ezen értékek között garantálja a megfelelő működést. A timer beállításánál az órajelet érdemes leosztani 80-1 értékkel, mivel ebben az esetben a timer számlálója $1\mu s$ elteltével növeli az értékét.

A szenzor működéséhez egy digitális kimenetet és egy timert használtam fel. A digitális kimenet szolgál arra, hogy kiadjuk a trigger impulzust, amivel elkezdődik a mérés. A timer pedig arra szolgál, hogy az Echo lábon kiadott jel hosszát mérjük. A

timernek van olyan beállítási lehetősége, hogy a mikrovezérlő egyik lábára kössünk egy jelet és a timer akkor kezdje el a számolást amikor felfutó élt detektál. Alapvetően ezt a beállítást alkalmaztam. A konfigurációban azt is megadtam, hogy ebben az esetben a mikrovezérlő megszakítás kezelője léptesse be a programot egy úgynevezett interrupt függvénybe. A szenzor működése ebben a függvényben lett implementálva. Ha felmenő élt detektál akkor timer értékét elmenti és a timer konfigurációját át állítja, hogy lefutó élt detektáljon. Ha újra beérkezünk ebbe a függvénybe, akkor az azt jelenti, hogy lefutó él detektálása történt, azaz a jelünk megérkezett a szenzor vevő egységébe. Ekkor ismét elmentjük az értéket és a két mért érték különbségéből tudunk távolságot számolni.

A távolság számításánál figyelembe kell vegyük azt, hogy milyen sebességgel terjed a hang a közegben. A távolság számításának képlete a következő:

$$Distance = \frac{Difference * 0.34\frac{cm}{s}}{2}$$

4.3 1. képlet Távolság számolása a kapott értékekből

A Distance a mért távolságot jelöli, a Difference a két eltárolt számláló érték különbségét, a 0.34 a hangsebesség levegőben. A kettővel való osztásra azért van szükségünk, mivel az ultrahang levegőben töltött idejét mérjük és a tárgy felé tartó és a visszaverődés utáni időt is mérjük. A számítások után a végső érték egy milliméterben lévő érték lesz, amit el kell tároljunk egy változóban, mivel ezt a későbbiekben el szeretnénk tárolni adatbázisban. [17, 18]

4.4 Távolság érzékelése lézeres alapú szenzorral

A lézeres alapú távolság érzékelésére VL53L0X szenzort használtam. A rendszerbe való beépítéshez már egy kész függvény könyvtárat alkalmaztam, amely elérhető az interneten. A kommunikációhoz az I^2C (Inter-Integrated Circuit) interfészt használja. A mikrovezérlő konfigurációjánál lehetőségünk van megadni a kommunikáció sebességét. A sebességnél ki kell válasszuk a Fast Mode lehetőséget, ami 400KHz es sebességet állít be. Ez az ajánlást a szenzor adatlapjában is olvashatjuk. A mérésnek az elve hasonló, mint az előző fejezetben említett ultrahangos szenzor, de ez a szenzor a lézer jel levegőben töltött idejéből számolja a távolságot.

A programba való integrálásához két fő függvényt használtam. Az egyik az inicializálást végzi. A függvényben érdemes először az XSHUT lábon keresztül újraindítani a szenzort. Ezt követően a használ függvény könyvtár segítségével inicializáljuk a szenzorunkat, amivel létre is hozzuk a kapcsolatot a mikrovezérlő és a szenzor között. A szenzor több egymás utáni lézer jelet is kiad és várja, hogy megérkezzen. Az érzékelési módból négy darabot olvashatunk az adatlapban. Alap mód, nagy pontosságú mód, nagy távolság mérési mód és gyors mérési mód. Az inicializáló függvényben a nagy pontosságú módot választottam, mivel nem időkritikus a mérésünk, ezért ez tűnt a legészszerűbbnek. A mérési módok között valójában annyi a különbség, hogy mennyi ideig várja a lézer jel visszaverődését. A szenzor megfelelő működése abban az esetben garantált, ha a tápfeszültsége 2.8V és 3.5V közé esik és a kőmérséklet -20°C és 70°C között van.

A szenzor inicializálás után a *readRangeSingleMillimeters()* függvénnyel tudjuk a távolságot mérni. Ahogy a függvény nevében is szerepel a mért távolságot milliméterben kapjuk meg. Miután kiolvastuk az értéket azt el is tároljuk és a későbbiekben bekerül az adatbázisba is. [19, 20]

4.5 Tömeg mérőcella

A tömeg mérésére a legelterjedtebb módszere a nyúlás mérő bélyegek használata. A nyúlás mérő bélyeget egy felületre ragasztva képesek vagyunk mérni a felület deformációját. A bélyeg egy vezetőréteg, amelynek a felület változására változik az ellenállása. Mivel az ellenállás változás nagyon kis mértékű ezért különböző erősítése módszereket és pontos mérést igényel a detektálása. A tömeg mérő cella mérőhíd elrendezésben van megvalósítva. A mérőcellát egy HX711-es modulra kötve használom, ami erre a célra lett kifejlesztve. A modul tartalmaz egy programozható erősítőt, amely 32, 64 és 128 erősítést tesz lehetővé az analóg jelen. Az erősítés után egy 24 bites analóg digitális konverter segítségével digitalizáljuk a jelet. A modulban megtalálható egy analóg feszültség szabályozó is, aminek az a szerepe, hogy a bemenő tápfeszültséget szabályozza. Így a modul tápfeszültsége 2.6V és 5.5V között lehet. A mikrovezérlővel való kommunikációja hasonlít az SPI interfészhez. Generálnunk kell bizonyos számú (kiválasztott erősítéstől függő) impulzust az SCK lábra és minden kiadott impulzusra egy értéket tudunk kiolvasni a DT lábon. Tehát az SCK egy órajelnek felel meg, míg a DT lábon keresztül történik az értékek kiolvasása.

A szenzor használatához már meglévő függvény könyvtárat használtam, ami nyíltan elérhető a Github oldalon. A fentiekben és az adatlapon leírt működési elvet implementálja. A mérőcella használatához először inicializálnunk kell azt. A könyvtárban leírt strukturából létre kell hozzunk egy példányt. Meg kell adjuk, hogy a mikrovezérlők lábai közül melyiket szeretnénk használni erre a célra. Majd megadjuk az erősítés mértékét. Ha ezzel végeztünk meghívjuk a *HX711_Tare()* függvényt ami beállítja az offsetet. Ez tulajdonképpen egy tárázás. Ha ezzel végeztünk akkor bármikor lekérhetjük a szenzor által mért eredményt.

A fent lévő publikus könyvtár nem teljes mértékig bizonyult megfelelőnek így a *HX711_Value* függvényben kis mértékű módosítást kellet végrehajtanom. A mérleg pontos használatához szükségünk van egy skálázásra. Ezt úgy tudjuk megtenni, hogy a mérleg értékét kiolvassuk üres állapotban, majd rá rakunk egy ismert tömeget és kiolvassuk az értéket. Végül az alábbi képlettel tudjuk megadni a skálázási együtthatót:

$$Skála\ érték = \frac{(Terhelve\ mért\ érték - Üresen\ mért\ érték)}{Terhelési\ tömeg\ [g]}$$

4.5 1. képlet Mérleg skálázásának számítása

Az offsetet a tárázási funkcióval tudjuk megkapni. A skálázási együttható meghatározása után már minden mérésnél használhatjuk ezt. A beolvasott értékből minden alkalommal ki kell vonjuk az offsetet és el kell osztanunk a már meghatározott skála értékkel. Az eljárás eredménye végül egy grammban meghatározott érték lesz. Abban az esetben amikor beüzemeljük a rendszerünket figyelnünk kell arra, hogy indulásnál a mérendő növény még ne legyen rajta a mérlegen, mivel akkor nem valós offsetet állítana be, ami durva mérési hibához vezetne. [21, 22, 23]

4.6 Hőmérséklet és páratartalom mérése

A környezet hőmérsékletének és páratartalmának mérésére egy DHT11 szenzort használok. Ennek a szenzornak előnye az, hogy elég jól ellenáll a környezeti hatásoknak, 3.3V és 5V tápfeszültséggel is üzemeltethető és az árához képest nagyon pontos mérésre képes.

A használatához függvény könyvtárat használok. A kommunikációt egy digitális láb segítségével és egy timerrel lehet megvalósítani. A timer azért kell, mert a

kommunikációban μs nagyságrendű ideig kell bizonyos értéken tartanunk a GPIO láb értékét. A timer konfigurációjánál 80-1 értékkel leosztjuk az alap órajelet. A kommunikációhoz a digitális portot váltogatnunk kell. Egyszer kimenetként, egyszer pedig bemenetként használjuk azt. A beolvasott adatból az első két byte felel a páratartalomért és a második két byte a hőmérsékletért. Az értékek beolvasásához csak a *readDHT11()* függvényt kell használjuk, mivel a szenzor kalibrálása nem szükséges. [24, 25]

4.7 Föld nedvesség érzékelése

A föld nedvességének egy kapacitív földnedvesség érzékelő szenzort alkalmaztam. Tápfeszültségnek 5V-ot választottam. Annak érdekében, hogy a szenzoron mért jel biztosan ne haladja meg a mikrovezérlő bemeneti feszültség maximumát, egy feszültség osztó kapcsolást építettem. A feszültségosztó kapcsolással a mikrovezérlő bemenetén lévő feszültség pontosan a fele annak, amit a szenzor kimenetén mérhetünk.

A szenzor kalibrálását úgy végezhetjük el, hogy meg mérjük a szárazon kiadott értéket, majd egy pohár vízbe merítjük és ez lesz a skála két végpontja. A skálázás számomra nem bizonyult egy jó módszernek, mivel nagyon eltérő eredményeket kaptam attól függően, hogy a helységben milyen volt a levegő páratartalma, ezért az analóg digitális konverter által mért értéket tárolom el az adatbázisba. Ez azért lehet egy jó megoldás, mert főként az érték változását és nem a pontos értéket szeretnénk tudni.

4.8 Színek érzékelése

A színek érzékelésére egy TCS3472 szenzort alkalmazok, ami az előző szenzoroktól eltérően az ESP32 mikrovezérlővel van összekötve. Kommunikációra az I^2C interfészt használja. Az Arduino környezet által lehetőségünk van mások által megírt könyvtárakat használni. Az alkalmazásomban az Adafruit által kiadott könyvtárat használom. A könyvtár használata igen egyszerű. Létre kell hozni egy példányt, amiben megadhatjuk a beállási időt és az erősítést, majd megkezdjük a kommunikációt a szenzorral. Ezek után már csak be kell olvassuk az értékeket. A szenzorral lehetőségünk van érzékelni a piros, zöld, kék, és a fehér értékeket. Mivel nekünk a fehér érték nem hordoz információt ezért azt nem használom fel.

5. Mikrovezérlőn futó programok rövid leírása

Az STM32 mikrovezérlőn futó program végzi a mérések nagy részét, ezért fontos, hogy megbízhatóan működjön. A program belépési pontja a *main()* függvény, amelynek az első részében a konfigurációban megadott inicializálások hajtódnak végre. Amikor ez megtörtént akkor inicializáljuk a szenzorjainkat a fentebb említett módokon. Szükségünk van egy UART kommunikáció megvalósítására is, mivel ezen keresztül fognak kommunikálni egymással a mikrovezérlők. Az inicializáláshoz tartozik az is, hogy UART-on keresztül küldünk egy „S” karaktert az ESP32 mikrovezérlő számára, amely ezt fogadja és visszaküldi az adatbázisban lévő eszközhöz tartozó mintavételezési időt. Ha ez sikeresen megtörtént akkor ezt el is mentjük egy változóba. A *main()* függvényben található egy végtelen *while()* ciklus ami a mikrovezérlő folyamatos futását biztosítja. Itt történik a mérési eredmények lekérdezése is. A mikrovezérlőn be kell konfigurálni a valós idejű órát, mivel ennek segítségével tudjuk időközönként elindítani a mérést. A mérések két esetben indulnak el, az egyik az, amikor letelt a mérési időköz, a másik pedig az, amikor azonnali mérést kezdeményezünk a weboldalon. Ekkor az ESP32 mikrovezérlő kap egy HTTP kérést és ennek hatására UART-on keresztül küld jelet az STM32 mikrovezérlőnek. A mérések végeztével az adatot UART-on keresztül elküldjük az ESP32 mikrovezérlőnek JSON formátumban.

Az ESP32 eszközön futó program nagyrészt a WiFi kommunikációért felelős. A WiFi kapcsolat létesítéséhez meg kell adjuk a WiFi hálózat SSID-jét és jelszavát. Ha ez is megtörtént akkor elindítunk egy HTTP szervert a mikrovezérlőn, amire majd az API küldi a kéréseket. Annak érdekében, hogy a server oldali API elérje statikus IP-címzést használok. A WiFi csatlakozás után kapcsolódunk a szín érzékelő szenzorhoz. A továbbiakban figyeljük az UART és a HTTP szerverre érkező adatokat. Abban az esetben, ha a HTTP szerver „/measure” végpontot hívjuk, akkor UART adatot küldünk az STM32 mikrovezérlőnek, ami ennek hatására azonnali mérést kezdeményez. A „/config” végpontra érkező kérés hatására pedig lekérdezzük az API-tól az adott mikrovezérlő mintavételezési idejét és UART-on továbbítjuk.

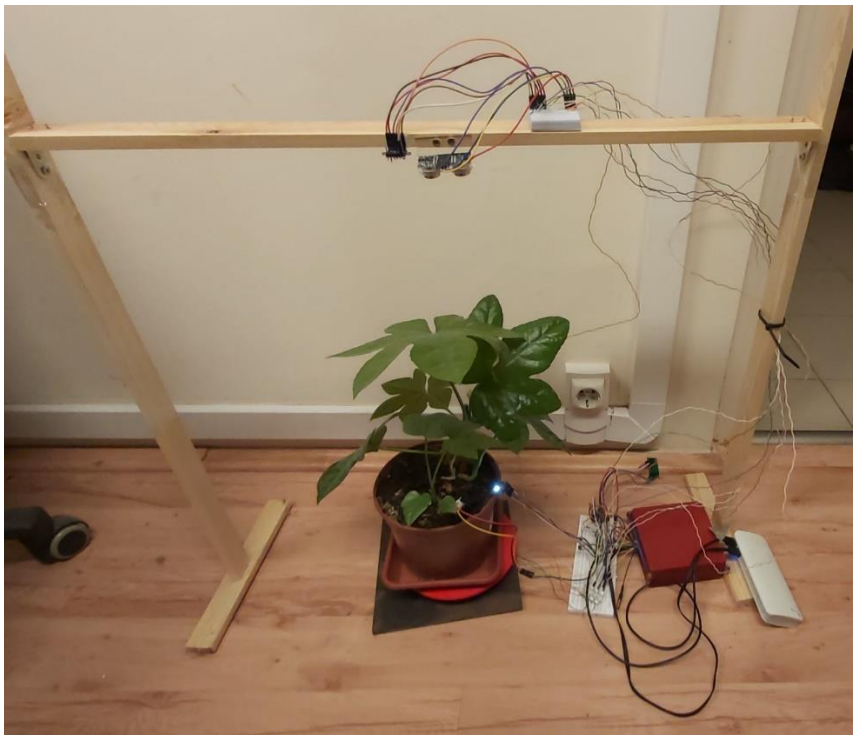
Az UART-on érkező jelek feldolgozásának két esete van, az egyik az, amikor egy ’S’ karaktert kapunk. Ebben az esetben ugyan az történik, mint a „/config” végpont hívásánál. A másik eset mikor adatot kapunk JSON formában. Ekkor elkezdjük a színek érzékelését. Ha ezzel végeztünk akkor a JSON formátumban kapott adatot kibontjuk és

hozzá fűzzük a mért szín adatokat, majd elküldjük az erre a célra létrehozott API végpontra.

6. Rendszer tesztelése

A rendszer tesztelésére nem volt lehetőségem valós környezetben, ezért kialakítottam egy saját tesztkörnyezetet. A saját tesztkörnyezetben arra törekedtem, hogy minél életszerűbb legyen a megvalósítás.

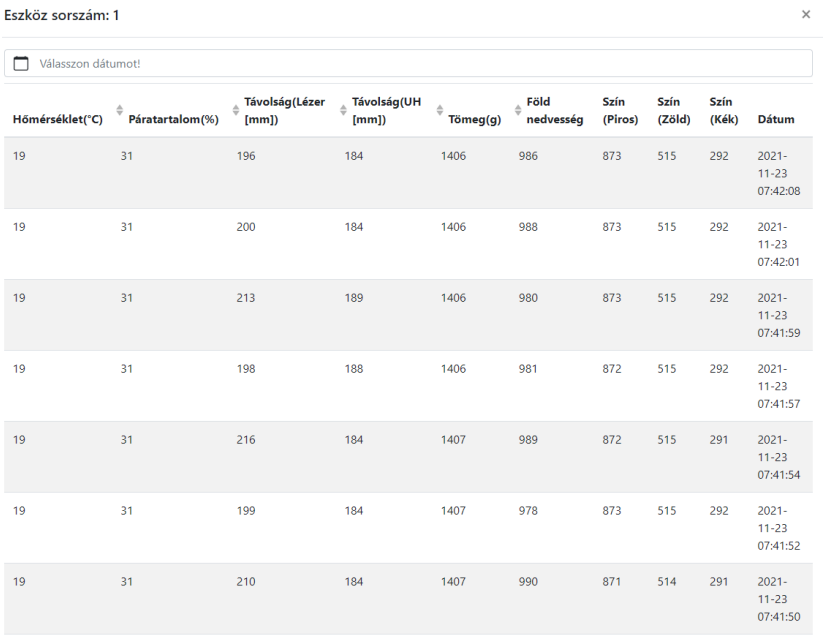
A szenzorok elhelyezéséhez készítenem kellett egy keretet, mivel a növény magasságának méréséhez a távolság mérésére alkalmas szenzorokat a növény felett kellett elhelyeznem. A tömeg mérésére alkalmas súly mérő cella a kis mérete miatt nem tartotta volna meg a növényt. Azért, hogy a tömeg mérését megvalósítsam terveztem egy tartót, amire a növény helyezem. Ezt 3D nyomtatással valósítottam meg. Az elvárásoknak megfelelően stabilan tartja a növényt.



6. 1. ábra Teszt környezet mérés közben

Az eredmények fedik a valóságot egy szenzor kivételével. Ez a szenzor a szín érzékelésére használt szenzor. A továbbiakban érdemes lehet ezt lecserélni egy jobb

minőségű szenzorra. A tesztelés során a mikrovezérlőről másodpercenként küldtem az eredményeket annak érdekében, hogy a kis változások is látszódjanak az eredményeken.


Eszköz sorszám: 1

Válasszon dátumot!

Hőmérséklet(°C)	Páratartalom(%)	Távolság(Lézer [mm])	Távolság(UH [mm])	Tömeg(g)	Föld nedvesség	Szín (Piros)	Szín (Zöld)	Szín (Kék)	Dátum
19	31	196	184	1406	986	873	515	292	2021-11-23 07:42:08
19	31	200	184	1406	988	873	515	292	2021-11-23 07:42:01
19	31	213	189	1406	980	873	515	292	2021-11-23 07:41:59
19	31	198	188	1406	981	872	515	292	2021-11-23 07:41:57
19	31	216	184	1407	989	872	515	291	2021-11-23 07:41:54
19	31	199	184	1407	978	873	515	292	2021-11-23 07:41:52
19	31	210	184	1407	990	871	514	291	2021-11-23 07:41:50

6. 2. ábra Mérési eredmények a weboldalon megjelenítve

7. Eszközök sérülésének elkerülése

Annak érdekében, hogy a mikrovezérlőket megvédjem a magas páratartalom okozta sérülésektől terveznem kellet egy dobozt, ami nem engedi be a párát. Erre a legalkalmasabb megoldásnak a 3D nyomtatást találtam. Mivel a Szegedi Tudományegyetem Műszaki Informatika Tanszékén rendelkezésre álltak a hozzá szükséges eszközök. A 3D modell megtervezéséhez az Autodesk Fusion 360 szoftvert használtam. A nyomtatáshoz nem elegendő ennek a szoftvernek a használata, megismerkedtem a Prusa Slicer szoftverrel, ami kifejezetten a Prusa 3D nyomtatókhoz lett fejlesztve. A doboz nyomtatásához a megjelölt forrás alapján a PETG anyag bizonyult a legmegfelelőbbnek az elérhető anyagok közül. A Prusa Slicerben lehetőségünk van betölteni anyagtól függő beállításokat. A nyomtatási beállításoknál az alapértelmezett beállításokon kis mértékű módosításokat kellett csináljak. A nyomtató fej hőmérsékletét 5°C meg kellett emeljem, annak érdekében, hogy a rétegek sokkal jobban egybe olvadjanak és ne legyenek rések, a filament adagolási sebességét pedig megemeltem 20%-kal. Ezen felül a rögzítéshez használt fülek nyomtatásához segédanyagot kellet használni, mivel a nyomtató nem képes alátámasztás nélkül bonyolultabb alakzatokat

nyomtatni. A nyomtató padon fekvő elemek a hőváltozás hatására hajlamosak meghajolni ezért egy úgynevezett brim réteget is hozzá kellett adjak a nyomtatáshoz, aminek következtében sikerült úgy kinyomtatni a doboz részeit, hogy nem látható rajtuk meghajlás.

Azon felül, hogy a mikrovezérlőnek tervezni kellett egy dobozt, tervezni kellett egy felületet, amit rögzíteni tudok a mérőcellára. Az elvárás az volt ezzel szemben, hogy elbírjon egy átlagos növényt. Mivel a mérőcella csak 5kg tömegig mér, ezért elégséges volt ezt a tömeget elbírja. A mérlegre tervezett felület PLA anyagból lett nyomtatva, ami végül olyan erősnek bizonyult, hogy elbírja a kívánt tömeget. Az általam tervezett 3D modellek a mellékletben elérhetőek. [26]

Összegzés

A feladatkiírásban szereplő feladatokat sikerült maradéktalanul megoldanom. Végeredményként egy olyan rendszer sikerült kiépítenem, amely alkalmas növények és környezetük monitorozására. A mért adatok elég jól fedik a valóságot és a rendszer ezeket az adatokat képes el is tárolni. A webes kezelőfelület teljes mértékben ellátja a rá bízott funkciókat, de van lehetőség további funkciók bevezetésére is. A szerver oldali alkalmazás megfelelőnek bizonyult arra a célra, hogy összekapcsolja az adatbázis, a weboldalt és a mikrovezérlőket. Az adatbázis alkalmas arra, hogy nagy mennyiségű adatot tároljunk benne és a felhőnek köszönhetően mindig és mindenhol elérhető. A továbbiakban érdemes lehet átgondolni, hogy milyen szolgáltatónál lenne érdemes az adatbázist futtatni, mivel különböző és folytonosan változó árazással rendelkeznek. A feladat elvégzése során sikerült alaposabban megismerkednem a szenzorokkal és a mikrovezérlők nyújtotta lehetőségekkel. Ezen felül kitapasztaltam, hogy hogyan lehet 3D modelleket tervezni, majd azokat kinyomtatni 3D nyomtató segítségével. A feladat kiírásnak megfelelően sikerült egy zárt dobozt terveznem, ami igényes megjelenésű.

A rendszert nem volt lehetőségem valós környezetben kipróbálni, de érdemes lenne megtenni ezt a lépést a jövőben. A tesztelés által összegyűjtött tapasztalatok alapján lehetne tovább fejleszteni a rendszert és beleépíteni azokat az észrevételeket, amik a tesztelési folyamat közben jönnek elő. Továbbá érdemes lehet egy olyan módszert kidolgozni, amivel le tudunk fedni egy nagy méretű üvegházat is kis eszközszámmal. Érdemes lenne a 3D nyomtatott dobozt huzamosabb időre magas páratartalmú környezetbe helyezni, hogy meg lehessen bizonyosodni annak hatékonyságáról.

Irodalomjegyzék

[1] MongoDB – Why use MongoDB: https://www.mongodb.com/why-use-mongodb (megtekintve: 2021.11.10.)

[2] MongoDB – Document Databases: https://www.mongodb.com/document-databases (megtekintve: 2021.11.10.)

[3] MongoDB – Non-relational databases: https://www.mongodb.com/databases/non-relational (megtekintve: 2021.11.10.)

[4] Dr. Balázs Péter – NoSQL adatbázis-kezelők: http://www.inf.u-szeged.hu/~pbalazs/teaching/nosql_adatbaziskezeles.pdf (megtekintve: 2021.11.10.)

[5] JSON: https://www.json.org/json-en.html (megtekintve: 2021.11.10.)

[6] MongoDB Atlas - Database: https://www.mongodb.com/atlas/database (megtekintve: 2021.11.10.)

[7] MongoDB Atlas - Multi-cloud Data Distribution: https://www.mongodb.com/cloud/atlas/multicloud-data-distribution (megtekintve: 2021.11.10.)

[8] NodeJS: https://nodejs.org/en/about/ (megtekintve: 2021.11.11.)

[9] npm: https://docs.npmjs.com/about-npm (megtekintve: 2021.11.11.)

[10] Express: https://expressjs.com (megtekintve: 2021.11.11.)

[11] Besant Technologies - What is Express.Js: https://www.besanttechnologies.com/what-is-expressjs (megtekintve: 2021.11.11.)

[12] Express - Install: https://expressjs.com/en/starter/installing.html (megtekintve: 2021.11.11.)

[13] JWT.io: https://jwt.io/introduction (megtekintve: 2021.11.12.)

[14] npmjs - jsonwebtoken: https://www.npmjs.com/package/jsonwebtoken (megtekintve: 2021.11.12.)

[15] Vue.Js - Introduction: https://vuejs.org/v2/guide/ (megtekintve: 2021.11.13.)

[16] BootstrapVue: https://bootstrap-vue.org (megtekintve: 2021.11.13.)

[17] ControllesTech - HCSR04 Ultrasonic sensor and STM32: https://controllerstech.com/hcsr04-ultrasonic-sensor-and-stm32/ (megtekintve: 2021.11.20.)

[18] Hestore - HC-SR04 adatlap: https://www.hestore.hu/prod_getfile.php?id=8286 (megtekintve: 2021.11.20)

[19] ST Community - ZVita: https://community.st.com/s/question/0D50X00009XkWI2SAN/ported-vl53l0x-pololu-code-to-stm32-always-times-out-cant-find-error (megtekintve: 2021.11.20.)

[20] Hestore – VL53L0X adatlap: https://www.hestore.hu/prod_getfile.php?id=9725 (megtekintve: 2021.11.20.)

[21] Gingl Zoltán – Elektronika I, Műszererősítők: http://www.inf.u-szeged.hu/~gingl/hallgatoknak/elektronika1/muszererositok.html (megtekintve: 2021.11.20.)

[22] Hestore – HX711 adatlap: https://www.hestore.hu/prod_getfile.php?id=9723 (megtekintve: 2021.11.20.)

[23] Github, freakone – HX711 függvény könyvtár: https://github.com/freakone/HX711 (megtekintve: 2021.11.20.)

[24] Github,, mesutkilic – DHT11-STM32-Library: https://github.com/mesutkilic/DHT11-STM32-Library (megtekintve: 2021.11.20.)

[25] Umang Gajera: Basics of Interfacing DHT11/DHT22 Humidity and Temperature Sensor with MCU: http://www.ocfreaks.com/basics-interfacing-dht11-dht22-humidity-temperature-sensor-mcu/ (megtekintve: 2021.11.20.)

[26] Prusa Knowledge Base – Watertight prints: https://help.prusa3d.com/en/article/watertight-prints_112324?fbclid=IwAR2OhDo9IPIq5D6XBh_5VfBZ28zSqip0-nPOydn6fZtK9fWeWgglokNv9PI (megtekintve: 2021.11.20.)

Nyilatkozat

Alulírott Bányai Balázs mérnökinformatikus MSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Műszaki Informatika Tanszékén készítettem, Msc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy diplomamunkámat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

2021.11.29.

Aláírás

Köszönetnyilvánítás

Köszönetet szeretnék mondani a témavezetőmnek, Dr. Mingesz Róbert Zoltánnak, aki sok segítséget nyújtott a témával és a 3D nyomtatással kapcsolatban. Illetve családomnak és barátaimnak, akik mindig támogattak egyetemi tanulmányaim alatt.