C:\Users\szarnyasg\Downloads\bme_logo_nagy.eps

**Budapesti Műszaki és Gazdaságtudományi Egyetem**

Villamosmérnöki és Informatikai Kar

Méréstechnika és Információs Rendszerek Tanszék

Elosztott, vezeték nélküli szenzorrendszer interaktív kezelőfelületekkel

Készítette

Papp Dominik Edvárd Konzulens

Scherer Balázs Attila

2023

Tartalomjegyzék

[Összefoglaló 6](#_Toc152868104)

[Abstract 7](#_Toc152868105)

[1. Bevezetés 8](#_Toc152868106)

[1.1. Motivációm 8](#_Toc152868107)

[1.2. A feladat rövid értelmezése 8](#_Toc152868108)

[1.3. Felhasználhatóság 8](#_Toc152868109)

[1.4. A szakdolgozat felépítésének rövid összefoglalása 9](#_Toc152868110)

[2. Rendszer architektúra 10](#_Toc152868111)

[2.1. Adatgyűjtő 10](#_Toc152868112)

[2.2. Szerver 10](#_Toc152868113)

[2.3. Adatbázis 10](#_Toc152868114)

[2.4. Web/Browser 10](#_Toc152868115)

[3. Alap alkotóelemek 11](#_Toc152868116)

[3.1. Programozási nyelvek 11](#_Toc152868117)

[3.1.1. C nyelv 11](#_Toc152868118)

[3.1.2. C++ nyelv 11](#_Toc152868119)

[3.1.3. Python nyelv 11](#_Toc152868120)

[3.1.4. Webfejlesztési nyelvek 11](#_Toc152868121)

[3.1.5. Webfejlesztési keretrendszerek 12](#_Toc152868122)

[3.1.6. FreeRTOS 13](#_Toc152868123)

[3.1.7. MySQL 13](#_Toc152868124)

[3.2. Verziókezelés 14](#_Toc152868125)

[3.3. Fejlesztőkörnyezetek, fejlesztőeszközök 14](#_Toc152868126)

[3.3.1. Visual Studio Code (VS Code) 14](#_Toc152868127)

[3.3.2. Node.js 14](#_Toc152868128)

[3.3.3. STM32CubeIDE 14](#_Toc152868129)

[3.3.4. TouchGFX 15](#_Toc152868130)

[3.3.5. MySQL Workbench 15](#_Toc152868131)

[3.3.6. CoolTerm 15](#_Toc152868132)

[3.3.7. Esptool.py 15](#_Toc152868133)

[3.4. Hardver 15](#_Toc152868134)

[3.4.1. STM32H750B-DK 15](#_Toc152868135)

[3.4.2. ESP8266-01 16](#_Toc152868136)

[3.4.3. SHT31 SEN0385 Szenzor 17](#_Toc152868137)

[3.4.4. Számítógépek 18](#_Toc152868138)

[3.5. Szoftver (szoftveres alkotóelemek) 18](#_Toc152868139)

[3.5.1. Adatbázis 18](#_Toc152868140)

[3.5.2. Adatgyűjtő szoftvere 18](#_Toc152868141)

[3.5.3. Szerver 19](#_Toc152868142)

[3.5.4. Weboldal 19](#_Toc152868143)

[4. Komponensek 20](#_Toc152868144)

[4.1. Adatgyűjtő 20](#_Toc152868145)

[4.1.1. ESP 20](#_Toc152868146)

[4.1.2. Szenzor 21](#_Toc152868147)

[4.1.3. STM32H750B-DK 22](#_Toc152868148)

[4.1.4. Szoftver 22](#_Toc152868149)

[4.2. Szerver 28](#_Toc152868150)

[4.2.1. Architektúra 29](#_Toc152868151)

[4.2.2. Könyvtár struktúra 29](#_Toc152868152)

[4.2.3. Közös, saját adatbáziskezelő segédkönyvtár 29](#_Toc152868153)

[4.2.4. Közös, saját Socket-kezelő segédkönyvtár 30](#_Toc152868154)

[4.2.5. TCP szerver 31](#_Toc152868155)

[4.2.6. Backend 38](#_Toc152868156)

[4.2.7. Algoritmus 40](#_Toc152868157)

[4.3. Adatbázis 40](#_Toc152868158)

[4.3.1. Tábla 40](#_Toc152868159)

[4.3.2. Csatlakozás 41](#_Toc152868160)

[4.4. Weblap 41](#_Toc152868161)

[4.4.1. Könyvtárak 42](#_Toc152868162)

[4.4.2. Segédfüggvény 43](#_Toc152868163)

[4.4.3. Komponensek 43](#_Toc152868164)

[4.4.4. Inicializás 53](#_Toc152868165)

[4.4.5. Build file-ok 54](#_Toc152868166)

[5. Komponensek közvetlen kapcsolatai 56](#_Toc152868167)

[5.1. Adatgyűjtő és Szerver 56](#_Toc152868168)

[5.2. Szerver és Weblap 56](#_Toc152868169)

[5.2.1. API 56](#_Toc152868170)

[5.2.2. Webserver 57](#_Toc152868171)

[5.3. Szerver és Adatbázis 57](#_Toc152868172)

[6. Komponensek közvetett kapcsolatai 58](#_Toc152868173)

[6.1. Adatgyűjtő és Weblap 58](#_Toc152868174)

[7. Értékelés 59](#_Toc152868175)

[7.1. Elemzés 59](#_Toc152868176)

[7.2. Továbbfejleszthetőség 59](#_Toc152868177)

[Irodalomjegyzék 60](#_Toc152868178)

Hallgatói nyilatkozat

Alulírott Papp Dominik Edvárd, 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, 2023. 12. 08.

Papp Dominik Edvárd

# Összefoglaló

A mai információ orientált világban az igény a vezetéknélküli eszközökre és az ezeket támogató szoftverekre folyamatosan nő. Számtalan alkalmazási területen előfordulhat, hogy az információ beszerzésére látszólag egyszerre van szükség, és ezen információkra több, fizikailag egymástól lényegesen távol elhelyezkedő pontokon tartanak igényt. Egy ilyen probléma megoldása változatos kihívásokat hordozhat magával az információkra való szükség gyorsaságának mértékétől, az információk mennyiségétől és minőségétől, valamint számtalan egyéb, előre be nem látható körülménytől függően.

Erre a problémára teljesen ideális választás a globálisan elérhető internethálózat felhasználása. Ma már interneten keresztül virtuálisan bármely két pont összeköthető a bolygón, gyorsan és hatékonyan, ami nem csak lényegesen megkönnyíti az adatátvitelt, de egy globális méretű rendszer esetén elősegíti az erőforrások földrajzi különválasztását kis- és nagytávú esetben egyaránt.

Az én alkotásom az említett problémakör megoldására lokális környezetben kíván egy alap rendszerarchitektúrát nyújtani korszerű eszközök segítségével. A probléma megoldását egy kisméretű adatgyűjtő rendszeren keresztül szeretném bemutatni, melynek mind a hardveres, mind a szoftveres komponensei korabeli technológiákat használnak. Az alkotóegységek könnyű skálázhatóságának köszönhetően továbbfejlesztéssel és személyre szabással nem csak konkrét kis és lokális problémák megoldására tehető alkalmassá, hanem más eszközök és eszközhálózatok bevonásával egy nagyméretű és többrétű információs rendszert is tud alkotni.

A fentiek megvalósítása nem csak mérnöki szempontból hasznos, de a személyes növekedésemhez is lényeges hozzátett, hisz egy ilyen megalkotása, majd szöveggé formálása komoly kihívást jelentett.

# Abstract

In today’s information-driven world, the escalating demand for wireless devices and their supporting software presents an ever-growing challenge. In numerous application areas the need may arise for the simultaneous acquisition of information and to transport it across vast physical distances. Addressing this challenge requires navigating issues of speed, volume, and quality of information, as well as unforeseen circumstances.

An optimal solution to this intricate problem lies in leveraging the globally accessible internet network. In today’s day and age, almost any two points on the planet can be seamlessly interconnected through the internet. This not only facilitates rapid data transfer but also enables the geographical distribution of resources in both short and long-distance scenarios within a global-scale system.

My creation endeavours to establish a foundational system architecture that tackles this challenge at a local level, employing cutting-edge tools. The intention is to showcase this solution through a small-scale data collection system, integrating both hardware and software components utilizing state-of-the-art technologies. The inherent scalability of these units allows for further development, adapting the solution not only for small local issues but also for the creation of large-scale, multifaceted information systems by incorporating additional devices and networks.

The realization of this project is not only beneficial from an engineering standpoint but also contributed significantly to my personal growth. The creation and articulation of such a project presented a substantial challenge, fostering a rich environment for learning and innovation.

# Bevezetés

## Motivációm

A motivációm a feladatra többrétű. A specializációm a beágyazott rendszerek világába vezetett be, ámde szerettem volna a webfejlesztés és az internet világával megismerkedni mindamellett, hogy a megszerzett beágyazott rendszerekhez kapcsolódó tudásomat felhasználom, fejlesztem és kamatoztatom. Szerettem volna továbbá számomra új, korszerű programnyelvekkel és technológiákkal megismerkedni. Célom volt az is, hogy ergonomikus felhasználói élményt tudjak biztosítani, kreativitásomat fel tudjam használni.

A fejemben ezeket a feltételeket összegezvén és az engem érdeklő ipari területekkel összevetve, a következő project ötlete fogant meg bennem.

## A feladat rövid értelmezése

A szakdolgozat célja egy olyan mérnöki alkotás elkészítése, amely képes egy távoli, a külvilággal fizikai összeköttetésekkel nem rendelkező szenzor állomás (adatgyűjtő) adatait helyben (weboldalon) megjeleníteni.

Ehhez négy egység szükséges:

1. Egy adatgyűjtő állomás, amely képes adatokat gyűjteni majd nem fizikai kapcsolaton keresztül azokat továbbítani, és egy kijelző segítségével felhasználói utasításokat végrehajtani.
2. Egy szerver, amely képes az adatgyűjtő állomás adatait kezelni, valamint képes a weboldal felé adatokat, illetve webfileokat továbbítani.
3. Egy weboldal, amely segítségével megtekinthetjük az adatokat.
4. Egy adatbázis, amely képes az adatgyűjtő adatait és más segítő célú adatokat tárolni

Ennek a négy egységnek a logikai kapcsolatait az 1. ábra szemlélteti.



1. ábra

## Felhasználhatóság

A szakdolgozatom munkáját egy alapváznak tekintem, amelyet számos területen fel lehet használni, –kisebb-nagyobb továbbfejlesztésekkel. A munka összköltségétől és az adott környezetben való gazdasági megfizethetőségétől most eltekintek, pusztán az ideális felhasználhatósága a szempont.

Felhasználható:

1. Okosotthonokban helyiségek szobahőmérsékletének, illetve egyéb tulajdonságainak a mérésére és továbbítására a központi számítógép felé. Ez előnyös lehet olyan esetekben, ahol fizikai UTP kapcsolat létrehozása már túl költséges opció volna.
2. Kültéri mérőállomásokon gyűjtött adatok továbbítására távoli központokba.
3. Folyamatok elemzéséhez szükséges adatok összegyűjtésére és továbbítására.

Tulajdonképpen minden olyan helyzetben felhasználható, ahol az adatokra nem (csak) helyben van szükség, hanem egy távoli ponton (is).

## A szakdolgozat felépítésének rövid összefoglalása

A szakdolgozat olvasása során, a továbbiak kerülnek bemutatásra, sorban:

1. A Rendszer átfogó architektúrája. Itt bemutatásra kerül az alkotás működése átfogólag, a legabsztraktabb szinten.
2. Az alkotáshoz felhasznált diszkrét, alap alkotóelemek, beleértve a fejlesztőkörnyezeteket és segédszoftvereket.
3. Az alap alkotóelemekből alkotott komponensek, az 1-es ábrán bemutatott négy egység.
4. Az előbb említett komponensek közötti közvetlen kapcsolatok. Pl.: hogyan kommunikál az adatgyűjtő állomás a szerverrel.
5. A közvetlen kapcsolattal nem rendelkező komponensek közötti közvetett, logikai kapcsolatok.
6. A szakdolgozat lezárása, értékelése.

# Rendszer architektúra

A rendszer architektúrája (2. ábra) első ránézésre bonyolultnak tűnhet, azonban kellően logikusan tagolt és több, könnyen kezelhető részegységre bontható.

Ebben a fejezetben be szeretném mutatni a rendszert a lehető legnagyobb absztrakcióval élve, ugyanis részletes tárgyalására számos későbbi fejezetben sor kerül.



2. ábra

## Adatgyűjtő

Az Adatgyűjtő egy szenzor-mikrokontroller-Wifi modul hármasból áll. Feladata az adat gyűjtése és továbbítása egy TCP szerver felé. Ez a kommunikáció a szerver felé egyirányú.

## Szerver

A Szerver egy TCP szerverből és egy Web Backendből áll. A kettő között kapcsolat nincs.

A TCP szerver adatokat fogad és elküldi egy adatbázis felé mentési kéréssel.

A Backend szolgál webserverként, azaz ki tud szolgálni egy weboldalt a megfelelő file-okkal és szolgál API -ként is. Az API képes adatokat lementeni és lekérdezni egy adatbázistól.

## Adatbázis

Az adatbázis képes több kapcsolaton keresztül kéréseket kiszolgálni.

## Web/Browser

A webes felület képes jelezni szükségét adatok kérése iránt, valamint azokat megjeleníteni formázottan, a felhasználó képernyőjén.

# Alap alkotóelemek

Ahogy az előző fejezet 2. ábráján látható, a project meglehetősen komplex. Emiatt sok és változatos technológiai eszközöket igényel a megvalósítása. A felhasznált technológiákat ebben a fejezetben szeretném bemutatni az olvasónak.

## Programozási nyelvek

A projectben több programozási nyelv is felhasználásra került. Ebben az alfejezetben szeretném összegyűjteni ezeknek az összességét, és röviden bemutatni azokat.

### C nyelv

A [C programnyelv](https://en.wikipedia.org/wiki/C_(programming_language))[1] minden elektronikus eszköz programozásának egyik alappillére. Jellegzetessége, hogy rendkívül hardverközeli programozást tesz lehetővé és mégis könnyebben kezelhető, mint az Assembly programozás. A C nyelvet sok területen használják, például operációs rendszerek fejlesztésében, beágyazott rendszerek programozásában és számos más alkalmazásfejlesztési területen.

### C++ nyelv

A [C++](https://en.wikipedia.org/wiki/C%2B%2B)[2] egy kiterjesztése és továbbfejlesztése a C programozási nyelvnek. Ebbe a nyelvbe az objektumorientált programozás elvei is beépültek. A C++ lehetővé teszi a programozók számára, hogy adatokat és funkciókat osztályokba csoportosítsanak, ami nagyobb szervezettséget és újrafelhasználhatóságot eredményez. Az objektum orientáltság egy kritikus összetevője a jól skálázható, komplex szoftvereknek.

A C++-nak sok alkalmazási területe van, például szoftvertervezés, játékfejlesztés, valós idejű rendszerek tervezése és egyéb alkalmazások fejlesztése.

### Python nyelv

A [Python](https://www.python.org/)[3] egy magas szintű, általános célú programozási nyelv, melyet Guido van Rossum kezdett el fejleszteni az 1980-as években és azóta rendkívül népszerűvé vált a programozói közösségben. A Python két fő vonzereje az olvashatóság és a könnyen tanulhatóság.

A Python széles körű alkalmazhatósággal rendelkezik és sokféle területen használják, beleértve a webfejlesztést (pl. Django keretrendszerrel), az adatelemzést (pl. NumPy és Pandas könyvtárakkal), a gépi tanulást (pl. TensorFlow és Scikit-learn), az automatizálást és még sok más területet. A nyelvnek erős és aktív fejlesztői közössége van, ami folyamatosan bővíti a Python ökoszisztémáját, valamint számos hasznos könyvtárat és eszközt kínál a programozók számára.

A Python a folyamatos fejlesztések miatt kényes a verzió típusra. Kényszerű megemlítenem, hogy a projectben mindenhol a Python 3.11.3.-as verziója van használva!

### Webfejlesztési nyelvek

#### HTML

A [HTML](https://en.wikipedia.org/wiki/HTML)[4] (HyperText Markup Language) egy szöveg alapú jelölőnyelv a weboldalak készítéséhez. Az alapötlete az, hogy az oldal tartalmát, struktúráját és formázását címkékkel határozzuk meg. Ezek a címkék olyan elemeket reprezentálnak, mint fejlécek, bekezdések, képek, linkek és egyebek. Ezen címkékhez, elemekhez, attribútumokat is hozzárendelhetünk, amelyek további információkat szolgáltatnak.

A nyelv kódja böngészők által értelmezett és megjelenített, így a felhasználók láthatják a formázott weboldalt. A HTML az alapja a weboldalaknak, és más technológiákkal, például CSS (Cascading Style Sheets) és JavaScript segítségével személyre szabhatjuk az oldal stílusát és funkcionalitását. Ez a nyelv az első lépés a webfejlesztésben, és a tartalom, valamint a struktúra meghatározására szolgál a weben.

#### CSS

A [CSS](https://en.wikipedia.org/wiki/CSS)[5] (Cascading Style Sheets) egy fontos eszköz a webfejlesztésben, amely segít a weboldalak kinézetének meghatározásában. Az ötlet mögötte az, hogy különválasszuk a tartalmat (HTML) és a stílust (CSS). CSS-szabályokban meghatározhatjuk a színeket, betűtípusokat, az elrendezést, és más stilisztikai tulajdonságokat. Ezeket a szabályokat elhelyezhetjük külön fájlokban, vagy közvetlenül a HTML-dokumentumban is.

A CSS alkalmazása nemcsak esztétikus megjelenést eredményez, hanem lehetővé teszi a reszponzív tervezést is. Ez azt jelenti, hogy az oldalak alkalmazkodnak különböző kijelzőméretekhez és eszközökhöz, például asztali számítógépekhez, táblagépekhez és mobiltelefonokhoz. A CSS segítségével egységes megjelenést érhetünk el a weboldalakon, valamint könnyen karbantartható és skálázható stílusokat hozhatunk létre. Ezek által a CSS elengedhetetlen eszköz a modern webfejlesztésben, amely segít a felhasználók számára vonzó és jól működő weboldalak készítésében.

#### JavaScript

A [JavaScript](https://en.wikipedia.org/wiki/JavaScript)[6] egy weboldalak interaktivitásának és dinamizmusának létrehozására használt programnyelv. A JavaScript a böngészők által támogatott, tehát a felhasználók eszközein fut anélkül, hogy külön szoftvert kellene telepíteni. Ez lehetővé teszi a weboldalak dinamikus, lokális frissítését és az interaktivitás bevezetését. A nyelvnek számos beépített funkciója van, amelyek lehetővé teszik a [DOM](https://en.wikipedia.org/wiki/Document_Object_Model)[7] (Document Object Model) manipulálását, eseménykezelést, AJAX hívásokat és még sok minden mást.

A JavaScript lehetővé teszi webalkalmazások fejlesztését, beleértve a játékokat, űrlapkezelést, adatkezelést, animációkat és még sok más funkciót. Gyakran használják más technológiákkal, például HTML-el és CSS-el, hogy teljes körű webalkalmazásokat hozzanak létre. A nyelv ökoszisztémája gazdag és számos keretrendszer, könyvtár és eszköz érhető el a hatékonyabb fejlesztés érdekében. Az egyszerűsége és sokoldalúsága miatt a JavaScript ma az egyik legfontosabb nyelv a webfejlesztés területén.

### Webfejlesztési keretrendszerek

A webfejlesztési keretrendszerek jelentős szerepet játszanak a modern webalkalmazások fejlesztésében. Ezek az eszközök elősegítik a hatékony és gyors fejlesztést, lehetővé teszik a könnyű újrafelhasználást és javítják a project skálázhatóságát. Nélkülük egy mai korszerű weblap belátható időn belüli elkészítése szinte elképzelhetetlen. A keretrendszerek szabványosított és bevált megoldásokat kínálnak a gyakori webfejlesztési kihívásokra, mint például az adatkezelés, a felhasználói felület tervezése, a biztonság és a teljesítmény optimalizálása.

#### React és JSX

A [React](https://react.dev/)[8] egy népszerű JavaScript könyvtár a felhasználói felületek ([UI](https://en.wikipedia.org/wiki/User_interface)[9], User Interface) építésére. A JSX egy kiegészítő szintaxis, amely lehetővé teszi a React komponensek deklaratív leírását. A React és a JSX segítségével webalkalmazásokat készíthetünk, amelyek dinamikusan reagálnak a felhasználói interakciókra. A React komponensek egységességet és újrafelhasználhatóságot biztosítanak, valamint könnyen integrálhatók más projektekbe. További előnye a React-nek, hogy egyoldalú weboldalak készíthetőek vele, ami olyan webalkalmazást jelent, amely egyetlen HTML oldalt használ a tartalom megjelenítésére, és a tartalom frissítése vagy betöltése során az oldal nem tölt be teljesen új oldalakat. Ehelyett dinamikus tartalomcseréket JavaScript és egy virtuális DOM segítségével végez. A React az egyik legnépszerűbb front-end keretrendszer, amelyet számos nagyvállalat és rengeteg fejlesztő használ a modern webalkalmazások fejlesztéséhez.

#### Bootstrap

[Bootstrap](https://getbootstrap.com/)[10] egy nyílt forráskódú CSS keretrendszer, amely segít az egységes és reszponzív webdizájn kialakításában. A Bootstrap tartalmaz előre elkészített stílusokat, komponenseket és JavaScript funkciókat, amelyek felgyorsítják a weboldalak tervezését és fejlesztését. A Bootstrap alapvetően lehetővé teszi a fejlesztők számára, hogy gyorsan és egyszerűen hozzanak létre modern, mobilbarát weboldalakat anélkül, hogy mélyebb CSS vagy JavaScript tudással rendelkeznének. Bootstrap-et gyakran használják front-end fejlesztés során és egyesítik a React vagy más keretrendszerekkel, hogy még hatékonyabbá tegyék a webalkalmazásaik fejlesztését.

#### FontAwesome

A [FontAwesome](https://fontawesome.com/)[11] egy népszerű ikonkészlet és webes ikonfont, amelyet a webfejlesztés során gyakran használnak ikonok weboldalakba és webalkalmazásokba illesztéséhez. Az ikonok különböző kategóriákban érhetők el, és segítségükkel könnyen hozzáadhatók olyan vizuális elemek, amelyek javítják a weboldalak és webalkalmazások felhasználói felületét.

### FreeRTOS

A [FreeRTOS](https://www.freertos.org/)[12] egy nyílt forráskódú valós idejű operációs rendszer (RTOS, Real Time Operating System), amelyet beágyazott rendszerekben használnak. Az RTOS olyan operációs rendszer, amely biztosítja a programok számára a valós idejű ütemezést, ami azt jelenti, hogy a rendszer képes szigorú időkorlátok mellett végrehajtani a feladatokat és eseményeket.

A FreeRTOS hatékony és kis memóriaigényű, így ideális választás olyan beágyazott rendszerekhez, ahol a rendelkezésre álló memória és erőforrások korlátozottak. Egy RTOS célja az alkalmazások strukturált szervezése és a prioritások kezelése, amelyek révén biztosítja, hogy a legfontosabb feladatok elsőbbséget élvezzenek. Emellett a FreeRTOS-t számos processzorra és mikrovezérlőre lehet „telepíteni”.

A FreeRTOS-hoz gazdag fejlesztői támogatás és dokumentáció is tartozik, amelyek segítik a hatékony alkalmazások fejlesztését. Az egyszerűsége és a valós idejű ütemezési képességei miatt, a FreeRTOS a beágyazott rendszerek tervezésében és fejlesztésében népszerű megoldás.

A FreeRTOS-nak több is verziója elérhető. Szakdolgozatomban azt a FreeRTOS verziót használom, amelyet egy [CMSIS-RTOS2](https://arm-software.github.io/CMSIS_5/latest/RTOS2/html/index.html)[13] API felülettel egészítettek ki. Ennek köszönhetően a fontosabb párhuzamos programozási eljárások operációsrendszer-függetlenné váltak.

### MySQL

A [MySQL](https://www.mysql.com/)[14] egy népszerű, nyílt forráskódú relációs adatbázis-kezelő rendszer és programozási nyelv, amelyet széles körben használnak az alkalmazások adatbázisának kezelésére. A MySQL teljes körű adatbázis-kezelési megoldás, amely támogatja a relációs adatbázisok tervezését és az SQL (Structured Query Language) használatát. Lehetővé teszi a felhasználók, jogosultságok és szerepkörök definiálását a biztonságos adathozzáféréshez és az adatintegritás megőrzéséhez.

A MySQL kiváló teljesítményt és skálázhatóságot nyújt, valamint támogatja az indexeket, a tranzakciókat és a replikációt. Összességében a MySQL a világ egyik legelterjedtebb adatbázis-kezelő rendszere, és számos alkalmazásban használják adatok hatékony kezelésére és tárolására.

## Verziókezelés

A [Git](https://git-scm.com/)[15] egy terminálból vezérelhető verziókezelő rendszer, amelyet fejlesztők és csapatok használnak szoftverfejlesztési projectek során az együttműködésre valamint a kódváltoztatások nyomon követésére egyaránt. A Git rendkívül hatékony és alkalmas választás a kisebb és nagyobb projektekhez egyaránt. A Git-nek elérthető webes felülete is, a [GitHub](https://github.com/)[16].

A fájlok változtatásai csak akkor kerülnek mentésre a verzióba (etalon kódbázisba), amikor a fejlesztő explicit módon megadja. Ez lehetővé teszi a precíz és tiszta verziókezelést, valamint a könnyű visszatérést a korábbi verziókhoz.

A szakdolgozatom számos szoftver fejlesztéséből állt, így elengedhetetlen eszköz volt a Git. Mivel a fejlesztés során több számítógépet is használtam, így a számítógépek között is könnyen tudtam hordozni a kódjaimat.

## Fejlesztőkörnyezetek, fejlesztőeszközök

A project több különböző programnyelv használatát tette szükségessé, amelyekkel való fejlesztéshez más-más fejlesztőkörnyezetek használatosak. Bizonyos folyamatok megkönnyítésére, feladatok ellátására, illetve programnyelvek felhasználásához fejlesztőeszközökre van szükség. Ezeket a fejlesztőkörnyezeteket és fejlesztőeszközöket gyűjtöttem össze és írok róluk ebben az alfejezetben, röviden.

### Visual Studio Code (VS Code)

A [Visual Studio Code](https://code.visualstudio.com/)[17] egy Microsoft által fejlesztett ingyenesen elérhető, nyílt forráskódú fejlesztőkörnyezet. A VS Code könnyen testre szabható, bővíthető, és számos kiegészítővel rendelkezik, amelyek segítik a kódszerkesztést és a projectfejlesztést. Ez egy népszerű választás a web- és szoftverfejlesztéshez, hisz számos programozási nyelvet és keretrendszert támogat.

### Node.js

A [Node.js](https://nodejs.org/en)[18] egy szerveroldali JavaScript környezet, amely lehetővé teszi a JavaScript programok futtatását a szervereken és a háttérben. A Node.js a Chrome V8 JavaScript motoron alapszik (Chrome böngésző JavaScript engine-je), és lehetővé teszi a JavaScript kód végrehajtását szerveroldalon is, nem csak a böngészőben. Ez egy nyílt forráskódú platform, amely széles körben használt különböző alkalmazások és webes szolgáltatások fejlesztésében.

Node.js segítségével lehet React projectet indítani, amelyhez szervert is indít a localhost-on a könnyebb fejlesztés érdekében. Továbbá lehet vele Bootstrap, FontAwesome és számos egyéb könyvtárat hozzáadni egy React projecthez.

Példakódként, a terminálból a következő képpen lehet kezelni:

npm i -g create-react-app // React telepítő telepítése

npm create-react-app app // app nevű React project létrehozása

cd app // app nevű projectkönyvtárba való belépés

npm start // React applikáció elindítása (webservert is indít)

npm i bootstrap // Bootstrap installálása a projectbe

npm i fontawesome // Fontawesome installálása projectbe

### STM32CubeIDE

Az [STM32CubeIDE](https://www.st.com/en/development-tools/stm32cubeide.html)[19] egy integrált fejlesztőkörnyezet, amelyet az [STMicroelectronics](https://www.st.com/)[20] fejlesztett ki az STM32 mikrovezérlőkön alapuló alkalmazások fejlesztéséhez. A szoftver tartalmaz eszközöket a kódírás, a debuggolás és az eszközkezelés támogatására. Az STM32CubeIDE használatával a fejlesztők egyszerűen hozhatnak létre és tesztelhetnek beágyazott szoftvereket. Támogatja a C, C++ kód fejlesztését, valamint telepíthető szoftverkomponensként elérhető a FreeRTOS az STM32H750B-DK fejlesztőkártyára.

### TouchGFX

A [TouchGFX](https://www.st.com/en/development-tools/touchgfxdesigner.html)[21] szintén az STMicroelectronics által fejlesztett, és egy nyílt forráskódú GUI (Graphical User Interface) keretrendszer olyan alkalmazásokhoz, amelyek érintőképernyővel működnek. A TouchGFX segítségével könnyen tervezhetők és fejleszthetők az alkalmazások grafikus felületei. A keretrendszer támogatja a modern, vonzó és reszponzív felhasználói élmény kialakítását különböző beágyazott eszközökön.

A szoftver teljes funkcionalitása a dolgozat írásakor még csak Windows operációs rendszeren volt elérhető, macOS és Linux rendszereken nem volt támogatott.

### MySQL Workbench

A [MySQL Workbench](https://www.mysql.com/products/workbench/)[22] egy népszerű MySQL adatbáziskezelő kliensalkalmazás, amely lehetővé teszi az adatbázisokhoz való kapcsolódást, a lekérdezéseket és az adatmanipulációt. A MySQL Workbench segítségével a felhasználók könnyen létrehozhatnak, módosíthatnak és törölhetnek adatbázisokat, valamint futtathatnak lekérdezéseket a táblák és rekordok manipulálására. A MySQL Workbench beépített funkciókat kínál a tranzakciók kezeléséhez, az adatbázisok tervezéséhez és az adatmodell létrehozásához is.

### CoolTerm

A [CoolTerm](http://freeware.the-meiers.org/)[23] egy egyszerű soros kommunikációs szoftver, egy [PuTTY](https://www.putty.org/)[24] alternatíva. A CoolTerm lehetővé teszi a soros porton keresztüli adatkapcsolatot, adatátvitelt és a kommunikációt különböző eszközökkel, például mikrovezérlőkkel vagy egyéb hardverrel. A PuTTY nem elérhető macOS rendszereken és a CoolTerm egy kompetens alternatívának bizonyult.

### Esptool.py

Az [esptool.py](https://docs.espressif.com/projects/esptool/en/latest/esp32/)[25] egy parancssori eszköz az Espressif ESP8266 és ESP32 WiFi mikrovezérlők programozásához és firmware-k frissítéséhez. A fejlesztők az esptool.py segítségével tudnak kommunikálni az ESP-eszközökkel és feltölteni vagy lecserélni a firmware-t. A szoftver gyakran használják a beágyazott IoT (Internet of Things) eszközök fejlesztése során.

## Hardver

A project megvalósításához fizikai eszközökre is szükség van. Ebben az alfejezetben a felhasznált fizikai, hardveres elemeket mutatom be.

### STM32H750B-DK

Az STMicroelectronics által gyártott [STM32H750B-DK](https://www.st.com/en/evaluation-tools/stm32h750b-dk.html" \l "overview)[26] (3. ábra) egy olyan fejlesztői kártya, amelyet a STM32 mikrovezérlőkkel történő alkalmazásfejlesztéshez terveztek. A kártya számos funkciót, hardvert, perifériát, szoftvert és beépített debuggert kínál mind a komplex, mind a változatos igényű projectek fejlesztéséhez.

A close up of a circuit board

Description automatically generated

3. ábra

#### Mikrovezérlő

A kártya középpontjában az STM32H750 mikrovezérlő található, amely egy erős és sokoldalú ARM Cortex-M7 32 bites processzormaggal rendelkezik. Az órajel sebessége a szoftveres beállítástól függ, de akár 480 MHz-ig is terjedhet. Ez a mikrovezérlő lehetővé teszi a magas teljesítmény igényű alkalmazások futtatását.

#### Kijelző

Az STM32H750B-DK egy színes, 480 x 272 pixeles felbontású LCD érintőképernyőt is tartalmaz, amely lehetővé teszi a grafikus felhasználói felületek kialakítását és tesztelését. Ennek a fejlesztéséhez az STMicroelectronics biztosít egy TouchGFX nevű szoftvert.

#### Csatlakozások

A kártya számos beépített csatlakozóval rendelkezik, ideértve az USB, Ethernet és Audió interfészeket. Ez megkönnyíti a külső eszközökkel való kommunikációt.

#### Szenzorok

Az STM32H750B-DK szenzorokat is tartalmaz, például gyorsulásmérőt és giroszkópot, amelyek segítségével érzékelhetjük a kártya mozgását.

#### Bővítési lehetőségek

A kártya számos bővítőhelyet kínál, például Arduino-illesztőknek megfelelő csatlakozókat, illetve egy úgynevezett [Fanout Board](https://www.st.com/resource/en/user_manual/um2695-stmod-fanout-expansion-board-for-stm32-discovery-kits-and-evaluation-boards-stmicroelectronics.pdf)[27] is tartozéka. A Fanout Boardon többek között található: UART, SCL, I2C interface és ESP-01 foglalat. A Board fejlesztői kártyához való csatlakoztatása nem úgy történik ahogy elsőre gondolná az ember, ugyanis nem csak az intuícióval megy szembe a módja, de a gyártó által biztosított képeken is rosszul van feltüntetve. Az 3. ábrán látható csatlakozási mód nem helyes. Vagy a kártyát, vagy a Fanout Boardot meg kell fordítani.

### ESP8266-01

Az [ESP8266-01](https://nurdspace.nl/ESP8266)[28] egy kis méretű, olcsó, de erős Wi-Fi modul, amelyet a Espressif Systems fejlesztett ki. Mivel képes kezelni a TCP/IP protokollt és apró méretű, nagyon népszerűvé vált az elektronikai és az IoT közösségben. Az ESP8266-01 számos alkalmazást támogat, például a szenzorok vezérlését, adatok továbbítását vagy akár távoli vezérlést Wi-Fi hálózatokon keresztül. A modul 8 kivezetéssel rendelkezik és UART protokollt használ kommunikációra, 115200-as baudratettel. A memória mérete változó. Az általam felhasznált eszközé 1MB.

#### Bekötése

A modul bekötését a 4. ábra szemlélteti, és a következő:

PIN1: Föld, PIN2: UART\_Rx (ESP ezen a lábon küld), PIN3: 3V3, PIN4: 3V3, PIN5: 3V3, PIN6: 3V3, PIN7: UART\_Tx (ESP ezen a lábon fogad), PIN8: 3V3

A diagram of a circuit board

Description automatically generated

4. ábra

Ez a bekötés használati üzemmódba helyezi a modult. Programozói üzemmódhoz más bekötés szükséges.

#### Programozás

A modul programozása eltérő huzalozást igényel. Programozható üzemmódba bootoláshoz a PIN5 lábat földre kell kötni mielőtt tápot kapna és csak utána biztosítani a feszültséget. Ebben az üzemmódban esptool.py, illetve Arduino IDE segítségével programozható.

### SHT31 SEN0385 Szenzor

A szenzor (5. ábra) hőmérséklet és relatív páratartalom mérésére egyaránt alkalmas. Kellően pontos és a mérési tartományai is tágak. Pontos adatok az [adatlapjában](https://dfimg.dfrobot.com/nobody/wiki/88b31350da4f54d00989c74c6fa392f7.pdf)[29], valamint emészthetőbb bemutató leírás a [Dfrobot WIKI](https://wiki.dfrobot.com/SHT31_Temperature_Humidity_Sensor_Weatherproof_SKU_SEN0385)[30] oldalán található meg.

A szenzorral I2C interfacen lehet kommunikálni legfeljebb 1MHz SCK órajel sebességgel. 3,3V - 5V feszültséggel optimálisan működtethető.

A close-up of a cable

Description automatically generated

5. ábra

### Számítógépek

A szoftverek fejlesztéséhez két számítógépet használtam. A fejlesztés döntő része macOS Ventura és macOS Sonoma operációs rendszereken történt, ezeken azonban a TouchGFX nem volt támogatva. A TouchGFX-szel történő fejlesztés Windows 10-en, egy külön számítógépen esett meg.

## Szoftver (szoftveres alkotóelemek)

Az előző Hardver c. fejezetben bemutatott elemek kezeléséhez, illetve a project egyéb komponenseinek megvalósításához különböző szoftverek megírására volt szükség. Ebben a fejezetben az általam írt szoftverek vannak összegyűjtve és átfogó funkcióik, felhasznált technológiáik által röviden bemutatva.

### Adatbázis

Az Adatbázis egy MySQL adatbázis, amely a MySQL Workbench segítségével könnyen manipulálható. Kapcsolat is létesíthető az adatbázissal, ha az adott programnyelven megvan a megfelelő könyvtár hozzá.

Az adatbázis parancsokkal is vezérelhető, úgynevezett *[query](https://www.techtarget.com/searchdatamanagement/definition/query" \l ":~:text=A%20database%20query%20is%20either,other%20forms%20of%20data%20manipulation.)*-kkal[31]. Ezek, és a megfelelő könyvtár segítségével távolról is módosítható az adatbázis.

### Adatgyűjtő szoftvere

A mikrokontrollert vezérlő szoftver két fő egységre osztható. Egyrészről kezelnie kell az érintőkijelzőt, másrészről adatokat kell beolvasnia és továbbítania. A kezdeti project-et a TouchGFX fejlesztőkörnyezetben kell létrehozni, mert így fognak a szükséges kijelzőkezelő kódok legenerálódni.

#### Főprogram

A főprogram FreeRTOS és taszkok segítségével látja el feladatait. A kódja STM32CubeIDE fejlesztőkörnyezetben íródott, C nyelven.

Feladatai között megemlítjük:

1. Perifériák inicializálása és kezelése.
2. Adatok beolvasása mérőeszközről majd azok megformázása.
3. Adatok továbbítása.

#### Touchgfx

Ez az egység ugyan a főprogram egyik eleme, mégis külön egységként kezelendő. Ennek az az oka, hogy nem csak egy külön nyelven, külön fejlesztőkörnyezet segítségével íródott, hanem egy másik nagyon konkrét feladatot is lát el: az érintőkijelzőt hivatott teljes mértékben kezelni.

C++ nyelven, a TouchGFX és az STM32CubeIDE fejlesztőkörnyezetek segítségével íródott. A kijelzőn megjelenítendő elemek felfoghatók objektumokként, így szinte evidens a C++ használata. A TouchGFX szoftver segítségével viszonylag egyszerűen lehet vizuális elemeket létrehozni és leprogramozni. Ezeket az elemeket a szoftver automatikusan C++ kóddá alakítja. Bizonyos funkciók megvalósításához szükség volt a generált kód kiegészítéséhez, amit STM32CubeIDE segítségével lehetett eszközölni.

A Touchgfx szoftveregység feladata a kijelzőn kapott érintések értelmezése, és a kijelzőn ezeknek megfelelő változások kirajzolása. Ezek mellett valamilyen módon a C nyelvű főprogram felé továbbítja a felhasználó által bevitt és felhasználandó adatokat.

### Szerver

A szervernek két nagyon fontos feladata van: TCP üzeneteket tudjon fogadni és HTTP kéréseket tudjon kiszolgálni. A szakdolgozat elején megadott felhasználási javaslatokból az is következik, hogy egyszerre több helyről is kaphat a szerver adatot, így azokat tudni kell párhuzamosan kezelni.

#### TCP server

A TCP szerver Python nyelven íródott, VS Code fejlesztőkörnyezetben.

Főbb feladatai:

1. TCP kapcsolatok létesítése több porton is, párhuzamosan.
2. TCP üzenetekből adatok kinyerése.
3. A kinyert adatok formázása és lementése adatbázisba.

Az adatbázishoz való hozzáférés miatt feltétlenül szükség van a [mysql-connector](https://pypi.org/project/mysql-connector-python/)[32] könyvtárra. A TCP kapcsolatok kezelése végett a [socket](https://docs.python.org/3.11/library/socket.html)-[33], a párhuzamosítás miatt pedig a [threading](https://docs.python.org/3/library/threading.html)[34] könyvtárra van még szükség.

#### Web backend

A Szerver szolgál még webszerverként és API-ként is. Webserver, mert képes kiszolgálni egy weboldal forrás filejait, és API, mert képes HTTP protokoll segítségével adatokat szolgáltatni, jelen esetben egy adatbázis adatait.

Mindkét funkcionalitás megvalósításához a [flask](https://flask.palletsprojects.com/en/3.0.x/)[35] könyvtárat használtam, amely rendelkezik beépített fejlesztői szerverrel. Production szervernek a [gunicorn](https://gunicorn.org/)[36] HTTP szervert, debug szervernek pedig a Flask beépített szerverét használtam.

Az adatbázishoz való hozzáféréshez itt is elengedhetetlen a mysql-connector használata.

### Weboldal

A weboldalnak egy fő feladata van, mégpedig, hogy adatokat tudjon megjeleníteni. Ennek eléréséhez a React könyvtárat használtam, számos alkönyvtárával együtt. A Weboldal képes API endpoint-okon adatokat kérni. A weboldal kiszolgálásáért a már tárgyalt Szerver felel. A weboldal esztétikájának megvalósításához a Bootstrap és a FontAwesome CSS keretrendszereket alkalmaztam, némi saját CSS kóddal kiegészítve.

# Komponensek

Ennek a fejezetnek a célja az eddigieknél egy absztrakciós szinttel feljebbről megismertetni az alkotóelemeket. Ebben a fejezetben bemutatásra kerülnek a nagyobb komponensek funkciói a lehető legnagyobb fogalmi és gondolati izolációval a többi nagyobb komponenstől.

## Adatgyűjtő

Ebben az alfejezetben az Adatgyűjtő architektúrája, protokolljai és működése kerül bemutatásra.

Átfogóan, az adatgyűjtő feladata a környezeti hőmérséklet mérése és a mérések továbbítása egy TCP szerver felé. Ehhez szükséges: mérés, internetkapcsolat és szoftveres adatfeldolgozás. Ezek kapcsolatát a 6. ábra szemlélteti.



6. ábra

### ESP

Az Adatgyűjtőn belül az ESP felel a külvilággal történő kommunikációért. Ahogy már korábban említettem, UART kommunikációval vezérelhető és TCP üzenetekkel képes az interneten információt közölni, továbbítani.

#### Vezérlés

Az ESP vezérlésére ideális választás az [AT-Command-ok](https://docs.espressif.com/projects/esp-at/en/release-v2.2.0.0_esp8266/index.html)[37] parancsai. Ez könnyű vezérelhetőséget biztosít, miszerint egyszerű sztringeket UART-on továbbítva, látszólag bonyolult folyamatok vihetők végbe. Hasznos tulajdonsága továbbá az AT-Commandok-nak, hogy visszajelzést biztosítanak a parancsok állapotáról, melyeket ugyan úgy, UART-on közölnek. Ez jellegzetesen a *„ready\r\n”*, *„OK\r\n”* és *„ERROR\r\n”* sztringeket jelenti.

#### Felprogramozás

Az AT-Commandokat támogató firmware-t esptool.py segítségével lehet az ESP-re rátölteni.

A feltöltendő firmware [innen](https://github.com/CytronTechnologies/esp-at-binaries/blob/main/Cytron_ESP-01S_AT_Firmware_V2.2.0.bin)[38] ingyenesen letölthető. Ugyan elérhetőek több parancsot is támogató firmwarek, azok nem megfelelőek. Az általam felhasznált modul memória mérete 1MB, azok a firmware-ek pedig nagyobb helyigényűek. Meglepően nagy kihívást jelentett egy alkalmas méretű firmware megtalálása.

Ha a modul programozói üzemmódban csatlakoztatva van a számítógéphez és elérhető az esptool.py, akkor a következő paranccsal tölthető fel a firmware:

esptool.py --chip auto --port /dev/tty.usbserialname --baud 115200 --before default\_reset --after hard\_reset write\_flash -z --flash\_mode dio --flash\_freq 40m --flash\_size 1MB 0x0 #PathToBinFile/Cytron\_ESP-01S\_AT\_Firmware\_V2.2.0.bin

, ahol */dev/tty.usbserialname* a csatlakoztatott programozó neve (cmd paranccsal való meghatározás Mac-en: *ls /dev/tty.\**) és *#PathToBinFile* a letöltött .bin file abszolút elérési útvonala. Ezek után nem programozói üzemmódban (PIN5: 3V3) való rebootolás után, AT-Command-ok segítségével vezérelhető is a modul.

#### Konfigurálás és csatlakozás

A modul konfigurálása WiFi-re és TCP protokollt támogató szerverre való csatlakozáshoz az AT-Commandokkal viszonylag egyszerű.

AT+CIPMODE=0\r\n // Normál adatátviteli üzemmódba helyezi a modult

AT+CWMODE=1\r\n // Kliens üzemmódba helyezi a modult, azaz kliensként fog csatlakozni az internetre

ATE0\r\n // Az ESP alapesetben visszaküldi a kapott parancsokat UART-on. Ez a parancs úgy konfigurálja az ESP-t, hogy ezt ne tegye.

AT+CWJAP=”SSID”,”PSW”\r\n // SSID nevű és PSW jelszavú wifi hálózatra rácsatlakozik

AT+CIPSTART=”TCP”,”IPADDR”, PORTNUM\r\n // TCP protokollt használó, IPADDR IP című host PORTNUM port-jára csatlakozik

#### Adatküldés

Az adatküldés is nagyon egyszerű. Első körben meg kell mondanunk hány byte-nyi adatot szeretnénk küldeni, majd lezáró *„\r\n”* nélkül közölni azokat. Például 4 byte esetén:

AT+CIPSEND=4\r\n

adat

Ezzel az *„adat”* utf-8 kódolású sztringet küldtük el a TCP host-nak.

#### Lecsatlakozás

A Wifiről való lecsatlakozás az

AT+CWQAP\r\n

paranccsal, a TCP szerverről való lecsatlakozás pedig az

AT+CIPCLOSE\r\n

paranccsal tehető meg. Ez utóbbi a [FIN](https://www.geeksforgeeks.org/tcp-flags/)[39] flag-et állítja be a TCP üzenetben.

### Szenzor

A Szenzor az Adatgyűjtőn belül a környezet hőmérsékleti és páratartalmi adatainak méréséért felel. A mérést tetszőleges időpontban el lehet végeztetni, szoftverből. Ennek kezelése a szolgáltatott driverrel meglehetősen egyszerű.

sht3x\_handle\_t sht31 = {&hi2c4, SHT3X\_I2C\_DEVICE\_ADDRESS\_ADDR\_PIN\_LOW};

sht3x\_read\_temperature\_and\_humidity(&sht31, &temperature, &humidity);

A fenti két kódsor felel a megfelelő használatért.

Az első sor a megfelelő handle összeállítását végezi az olvasáshoz. A *hi2c4* a mikrokontroller által használt 4-es I2C interface handle-je, a második paraméter pedig egy, a driverbe beépített konstans.

A második kódsorral lekérdezhetőek az adatok, amelyek a *temperature* és *humidity* float változókban lesznek elérhetők.

### STM32H750B-DK

A kívánt funkciók eléréséhez némi módosításra van szükség a mikrokontrolleren. Ezek közé sorolom a fizikai és szoftveres módosításokat is. Az adatgyűjtéshez ugyebár az ESP-t, az érintőkijelzőt és a szenzort is használni kell, –a specifikáció szerint. Ezeknek a használata bizonyos előfeltételekhez van kötve, illetve elővigyázatosan kell őket kezelni.

#### Solder bridge-k

Mind a Szenzor, mind az ESP a Fanout bővítőkártyát használja. Az ESP használatához az UART interface-t ki kell vezetni a bővítőkártyára. Alapesetben nincs kivezetve, de solder bridge-ek átforrasztásával ez módosítható. A 7. ábra a mikrokontroller Fanout bővítőkártya foglalatának interface-kivezetési ábrája. Az ábra szerinti SB13-16 és SB11-12 solder bridge-ek átforrasztásával kivezethető az UART. A 4-es I2C interface alapból ki van vezetve, ehhez egyéb hardveres módosítás nem szükséges. A pontos logika a 7. ábráról leolvasható.

A diagram of a computer component

Description automatically generated

7. ábra

#### I2C

A mikrokontroller érintőkijelzőjét a TouchGFX által generált kód a 4-es I2C interfacen éri el, ami ugyan az a belső I2C busz, ami a bővítőkártyára ki van vezetve és amin a szenzor elérhető. Ez hibát okozhat abban az esetben, ha a buszt többen, egyszerre szeretnék használni. Ezt elkerülendően az erőforrást mutex-szel kell védeni. Ezt a szoftveres rész meg is teszi.

#### UART interrupt

Mivel az ESP UART-ot használ a kommunikációra és az általa közölt információkra szükség van, azokat valahogy fogadnunk kell. Erre optimális megoldás az UART Rx lábára interrupt-ot beállítani. A beállítás elvégzése ebben az esetben is szoftveres.

### Szoftver

Az Adatgyűjtő szoftvere felel az összes komponensének összehozataláért. Kulcsfontosságú a perifériák és az adatok megfelelő kezelése.

A 8. ábra mutatja a szoftver egészét. Első ránézésre bonyolultnak tűnhet, ámde a következő alfejezetekben részletesen tárgyaljuk és emészthető darabkákra bontjuk. A program architektúrája a következő elvet követi: A Touchgfx vezényli a Főprogramot a FreeRTOS szolgáltatásai segítségével. Ez a mondat később több értelmet nyer majd.

8. ábra

#### Főprogram

A főprogram C nyelven íródott az STM32CubeIDE feljesztőkörnyezetben. A megfelelő működés eléréséhez szükségszerűen felhasználásra kerültek a FreeRTOS által nyújtott szolgáltatások, mint pl: mutex-ek, queue-k, task-ok stb. A program alapvetően task-okba szervezett. A FreeRTOS-t az alapbeállításaival használom, melyek közül talán a két legfontosabb: 1000-es Tick Rate Hz-ben, preemptív ütemezéssel. Ebben a fejezetben részletesen bemutatom a task-okat és azok szerepeit. A 9. ábra mutatja pontosan mivel is foglalkozik ez az alfejezet, mi is számít a 8. ábrából a főprogramnak.



9. ábra

##### Initialise board

Ez a program belépési pontja és az egyik nem task kódrészlet. A mikrokontrollert inicializálni kell. A port-okat, lábakat stb. megfelelő állapotba kell helyezni, az ütemezőt elindítani, a változókat, semaphore-okat stb. létre kell hozni. Az ehhez szükséges kódot a fejlesztőkörnyezet automatikusan legenerálja a beállítások alapján. Az Adatgyűjtő a Fanout board-on használja a UART és I2C interface-t. A processzor lábait ezeknek megfelelően kell beállítani a fejlesztőkörnyezetben és a [Fanout user manual](https://www.st.com/resource/en/user_manual/um2695-stmod-fanout-expansion-board-for-stm32-discovery-kits-and-evaluation-boards-stmicroelectronics.pdf)-ja[27] és a [mikrokontroller user manual](https://www.st.com/resource/en/user_manual/um2488-discovery-kits-with-stm32h745xi-and-stm32h750xb-mcus-stmicroelectronics.pdf)-ja[40] segítségével. Az érintőkijelző használatához a beállításokat a TouchGFX automatikusan legenerálja és további módosítást nem igényel.

##### SetUpESP

Ez a task nem vár semmilyen task lefutására. Az ütemező indítása után azonnal futásra kész. Feladata előkészíteni az ESP-t a majdani csatlakozásokra. Először újraindítja, majd beállítja, hogy az ESP ne küldje vissza a kapott parancsokat. Ezután alkalmassá teszi TCP és Wifi kapcsolat létesítésére. Lásd: Konfigurálás és csatlakozás alfejezet.

Végső soron az *ESP basic setup finished* event flag-gel jelzi a sikeres futást és felfüggeszti saját magát. Futására csak egyszer van szükség.

##### ConnectToWifi

A task egy dolgot csinál, mégpedig egy adott SSID-jű és jelszavú Wifi hálózatra rácsatlakoztatja az ESP-t. Az ehhez szükséges adatokat változókból nyeri ki. Futásához két event flag beállítására van szükség: *ESP basic setup finished* és *ESP Wifi connect*. Ezután csatlakozik a Wifi-re, majd beállítja a *Wifi connected* event flag-et. Lásd: Konfigurálás és csatlakozás alfejezet.

##### ConnectToServer

Ez a task a TCP szerverre való csatlakozást kezeli. Adott IP című szerver adott port-jára csatlakozik az ESP-n keresztül. Az ehhez szükséges adatokat változókból nyeri ki. Futásához két event flag beállítására van szükség: *Wifi connected* és *ESP Server connect*. Ezután csatlakozik a TCP szerverre és beállítja a *Server connected* event flag-et. Lásd: Konfigurálás és csatlakozás alfejezet.

##### ReadTempAndHumidity

Ez a task hivatott a méréseket elvégezni adott időközönként. Futásához a *Server connected* event flag beállítására van szükség, majd végtelen ciklusban fut. Először elvégzi a mérést a szenzoron keresztül és 2 queue-ba helyezi az adatokat, hőmérséklet és páratartalom sorrendben. A 2 queue: *TouchgfxTempAndHumid* és *TempAndHumid.* Az előbbi, a Touchgfx felé szolgáltatja az adatokat, az utóbbi a főprogram számára. Ezek után *Data frequency* ideig felfüggeszti a futását és kezdni a mérési folyamatot elölről. A *Data frequency* változóban milliszekundum mértékegységű számértékre számít.

##### SendDataWithESP

A task feladata a mért adatokat továbbítani a TCP szerver velé. Futásának egy előfeltétele van: a *Server connected* event flag beállítása. Ha ez a feltétel teljesült, végtelen ciklusban futtatja a következő algoritmust: Kinyeri az adatokat a *TempAndHumid* queue-ból, megformázza azokat és továbbítja őket a Szerver felé. Lásd: Adatküldés alfejezet.

A formátum a következő: „int float floatq”. Az első *int* egy egyedi azonosító, a két *float* pedig a két mért adat. A *„q”* egy lezáró karakter.

Mivel az adatküldés 2 UART használatból áll, ezért az semaphore-ral van védve, hogy ne történjen UART használat a két parancs között, hisz ez hibát okozna.

##### Disconnect

A task, ha a *Disconnect request* event flag beállítását érzékelte, egyszeri lefutással futtatja kódját. A task célja megbontani a kapcsolatokat és leállítani a méréseket. Az ESP-t lecsatlakoztatja a TCP szerverről és a Wifi-ről. Lásd: Lecsatlakozás alfejezet. A kapcsolatok megbontása után a ReadTempAndHumidity és SendDataWithESP task-ok futását felfüggeszti.

##### UART\_RXCallback

Az UART\_RXCallback nem egy task, hanem egy callback függvény. Akkor fut le, ha az UART Rx lábán adat érkezett. Ekkor a kapott adatot egy *RxDataChar* queue-ba helyezi.

##### ReadESP

Ennek a task-nak a futása az előzőekkel ellentétben nem feltételhez kötött és szabadon, végtelen ciklusban fut. Az *RxDataChar* queue-ban szereplő adatokat dolgozza fel, azaz az ESP-től kapottakat. A task egy semaphore-al védett *expectedESPResponse* sztringgel hasonlítja össze minden új kapott karakter után az eddigieket. Ha megegyezik a kettő, akkor azt a *Response valid* event flag-gel jelzi. Ha az *„ERROR\r\n”* sztringet kapja, az az hiba történt a ESP-n, akkor azt az *ESP Error* event flag-gel jelzi.

##### bool send\_and\_receive(char\* send, char\* receive, bool useSemaphore)

A send\_and\_receive egy segédfüggvény az ESP-vel való kommunikációhoz. Ez az egyedüli programrész, amely közvetlenül eléri az UART interface-t. Az UART használatához a HAL könyvtárat veszi igénybe.

A paraméterek között megadható, hogy milyen karaktersorozatot küldjön el és milyen karaktersorozatot várjon válaszként. A *useSemaphore* paraméterrel azt lehet beállítani, hogy a függvény magától használja-e az UART semaphore-t, vagy sem. A semaphore-al védett *expectedESPResponse* sztringet ez a függvény állítja be a *receive* paraméter alapján.

A függvény 5000 milliszekundumot vár a *Response valid* event flag-re. Ha nem kapja meg, akkor megpróbálja még egyszer az elküldést. Ha így sem sikerül 5000 milliszekundum után helyes választ kapni, akkor hibát feltételez és beállítja az *ESP Error* event flag-et.

Sikeres kommunikáció esetén igazzal tér vissza, egyébként hamissal.

##### Default

Ez a task egyetlen egy dolgot csinál, mégpediglen az *ESP Error* event flag-re vár. Ha azt valaki beállítja, akkor az egész mikrokontrollert szoftveresen újraindítja.

#### Touchgfx



10. ábra

A Touchgfx a mikrokontroller érintőkijelzőjének kezeléséért felelős. Átfogóan a 10. ábra szemlélteti működését, amely a 8. ábra része. A megfelelő működés eléréséhez szükségszerűen felhasználásra kerültek a FreeRTOS által nyújtott szolgáltatások, mint pl: mutexek, queue-k stb. Fontos továbbá megemlíteni, hogy egy darab task-ként fut a Touchgfx. Más lehetőséget nem kínált az STM32 szoftvere (TouchGFX). Ez sajnos jelentősen lecsökkentette az egy félév alatt implementálható funkciók számát, ugyanis egy szoftver sok állapot eltárolását igényli, amit könnyű felváltani task-ok felhasználásával. Task-ok hiányában állapotgépesen lehetett volna megoldani a hiányzó funkciókat, de így viszont nem csak értelmét vesztené az operációs rendszer használata, hanem a fejlesztési idő is megsokszorozódott volna. Az idő szűkére való tekintettel, csak az operációs rendszer segítségével észszerűen implementálható funkciók kerültek lefejlesztésre.

Ez a szoftverkomponens C++-ban íródott. A TouchGFX ezen a programnyelven generál egy alapkódot abból, amit a TouchGFX vizuális felületén megadtunk. Egy task-ba generál mindent és automatikusan megoldja a C és C++ közötti fordítási problémát, így ezzel nem kell foglalkozni.

A Touchgfx task-ot 3 képernyőre lehet bontani: ScreenWelcome, ScreenSettings, ScreenHome.

##### ScreenWelcome

Ez a képernyő az első, ami indítás után megjelenik. Pusztán annyit csinál, hogy egy homályosítási átmenettel (fade-in, fade-out) egy üdvözlő üzenettel üdvözli a felhasználót. Utána rögtön vált a ScreenSettings képernyőre.

##### ScreenSettings

A ScreenSettings képernyőn tudja megadni a felhasználó egy [érintőkijelzős billentyűzet](https://controllerstech.com/touchgfx-7-how-to-implement-on-screen-keyboard/)[41] segítségével a TCP kapcsolat létesítéséhez szükséges konfigurációs paramétereket, melyek: egy wifi hálózat SSID-je és jelszava, a TCP szerver IP címe és portszáma. Ezen kívül még meg tudja adni milliszekundumban, hogy milyen gyakorisággal küldjön adatokat az Adatgyűjtő.

A képernyő attól függően hogy melyik képernyőről érkezik ide a felhasználó, más-más funkciókat lát el.

Ha a ScreenWelcome-ról érkezik, akkor csatlakozni tud a *Connect* gombbal a megadott paraméterű Wifi-re és szerverre. A *Connect* gomb megnyomása elmenti a bevitt adatokat és beállítja a csatlakozást jelző event flag-eket.

Ha a ScreenHome-ról érkezik a *Disconnect* gomb megnyomásával, akkor újracsatlakozni tud, ugyanis megjelenik a *Reconnect* gomb*.* A *Reconnect* gombnak a *Connect* gombhoz hasonló a hatása. Annyi a töbletfunkciója, hogy bizonyos felfüggesztett task-ok futását folytatja.

Ha a ScreenHome-ról a *Settings* gomb megnyomásával érkezik, akkor a megjelenő *Home* gombbal visszajuthat a ScreenHome-ra.

##### ScreenHome

A ScreenHome-on folyamatosan szemmel követhetőek a mért adatok. Az adatokat képernyő kódja a *TouchgfxTempAndHumid* queue-ból nyeri ki. Az erre a képernyőre való eljutás a ScreenSettings képernyőn történő csatlakozás után automatikusan megtörténik, illetve a ScreenSettings képernyőről a *Home* gombbal is el lehet jutni. A *Settings* gomb visszaviszi a felhasználót ScreenSettings képernyőre. A *Disconnect* gomb is hasonló funkciót lát el, de a képernyőváltás előtt még a *Disconnect* *request* event flag-et beállítja.

#### A Főprogram és a Touchgfx kommunikációja



11. ábra

A könnyebb átláthatóság kedvéért, a két szoftveregység közötti kommunikációt külön fejezetben, itt tárgyalom. A programban használt kommunikációs formákra tekinthetünk amolyan interfészként is, amely a Főprogramot vezényli. Az értelmezésben a 11. ábra kíván segítséget nyújtani.

A következőket a Touchgfx szemszögéből kell értelmezni.

##### Connection event flags

Ez két event flag-re bomlik: *ESP Wifi connect* és *ESP Server connect*. Ez a két flag végzi a Wifi-re és a szerverre való csatlakozást. Ezeket a *Connect* és a *Reconnect* gombok adják ki csatlakozási kérelemként.

##### Save configuration

Ez műveletek sorozata, amely elmenti a felhasználó által bevitt adatokat, melyek: Wifi SSID és jelszó, Szerver IP címe és port száma és adat gyakoriság (azaz milyen gyakran küldődjön adat). Ezek a műveleteket is a *Connect* és a *Reconnect* gombok hajtják végre, még a csatlakozási kérelem jelzések előtt.

##### Disconnect request

Ennek az event flag-nek a beállításával azonnali lecsatlakoztatás végezhető a Wifi-ről és a szerverről, melyet a *Disconnect* gomb ad ki.

##### Resume

A Resume újracsatlakozás (*Reconnect* gomb megynomása) esetén folytatja az adatküldést.

##### TouchgfxTempAndHumid

A Főprogram ezen a queue-n keresztül juttatja el a mért adatokat a Touchgfx felé.

## Szerver

A Szerver szerepe a projectben egyszerű, de talán a legfontosabb. Ahogy a Rendszer architektúra fejezetben bemutattam, a Szerver az összekötő kapcsa minden komponensnek. A Szerver teljes kódja Python nyelven íródott és több könyvtárat is felhasznál. Emlékeztetésképpen: A Szerver célja TCP protokollon keresztül adatokat fogadni és webfileokat- és adatokat kiszolgáltatni. A Szerver képes egyszerre több kapcsolatot is kezelni, és nagy előnyben részesíti a párhuzamos programozás elveit. Ez jelentősen növeli a program érthetőségét és olvashatóságát, ami megkönnyítette a leprogramozását.

### Architektúra

A Szerver architektúráját a 12. ábra szemlélteti. A kívánt funkciókért két komponens felel: TCP szerver és Backend. Az ábrából látszik, hogy a TCP szervernek és a Backend-nek van egy közös erőforrása: az Adatbázis. A Backend továbbá két részre osztható: Webszerverre, ami a webfájlokat-, API-ra, ami az adatokat szolgáltatja ki.



12. ábra

### Könyvtár struktúra

Server/

|-- \_\_ init\_\_.py

|-- main.py

|-- backend/

|-- \_\_init\_\_.py

|-- web\_backend.py

|-- common/

|-- \_\_init\_\_.py

|-- db\_handling.py

|-- my\_socket.py

|-- TCP\_server/

|-- \_\_init\_\_.py

|-- data\_handling.py

|-- my\_port.py

|-- TCP\_server.py

A fenti kódszerű leírás reprezentálja a Szerver könyvtárának struktúráját. Sok helyen megjelenik az *\_\_init\_\_.py* file. Így lehet jelezni a Python felé, hogy kezelje az adott könyvtárat úgynevezett csomagként. Ez segít a file-ok importálásában.

### Közös, saját adatbáziskezelő segédkönyvtár

Ez a könyvtár a project számára egy kezelőfelületet kíván nyújtani az adatbázis kezeléséhez (*db\_handling.py*). Ez az alfejezet csak a project megértéséhez szükséges távlatból mutatja be a könyvtárat, nem kíván dokumentációt nyújtani a kódhoz.

#### Globális változók

HOST = "localhost"

USER = "pythonUser"

PASSWORD = "pythonUser"

DATABASE = "weather"

Ez a 4 változó tartalmaz minden információt, ami egy adatbázishoz való csatlakozást lehetővé tesz. A könyvtár ezeket a változókat közvetlen nem használja fel. A célja inkább a könyvtár felhasználói számára beépített változókat biztosítani egy bizonyos adatbázishoz. A továbbfejleszthetőség szemüvegén keresztül nézve sokkal logikusabb, ha a paramétereket a felhasználó adhatja meg –például csatlakozáskor–, hisz ha több adatbázishoz szeretne csatlakozni és csak a beépített változókat lehet használni, akkor csak egy adatbázist tudna elérni.

#### connect\_to\_db(host, user, password, databse)

Ahogy a neve is sugallja, ez a függvény segít az adatbázishoz való csatlakozásban. Paraméterekként az adatbázis paraméterei adhatóak meg. Sikertelen csatlakozás esetén *None* a visszatérési értéke, sikeres csatlakozás esetén pedig a csatlakozást jelképező *connection* és az irányítást lehetővé tevő *cursor* változókkal tér vissza.

#### make\_insert\_query(id, temp, humidity, dataDate)

Ez a függvény visszatér egy MySQL insert (beillesztési) query-vel, amit a paraméterként kapott változókból állít össze. Ezeknek a paramétereknek az indokoltsága később válik világossá.

#### send\_query(cursor, connection, query)

Ezzel a függvénnyel egy MySQL query elküldhető az adatbázisnak végrehajtásra. Az adatbázist a *cursor* és *connection* változók reprezentálják.

#### disconnect\_db(cursor, connection)

Ahogy a *connect\_to\_db* az adatbázisra rákapcsolódik, úgy ez a függvény megbontja a kapcsolatot.

### Közös, saját Socket-kezelő segédkönyvtár

Ez a könyvtár (*my\_socket.py*) socket-ek kezelését kivánja leegyszerűsíteni.

#### get\_local\_ip()

Visszaadja annak a gépnek az alhálózatbeli IP címét amin jelenleg fut a kód.

#### set\_up\_socket(maxRetrys, port)

Egy socketet nyit a gép alhálózatbeli IP címén, a *port* portszám alatt. A *maxRetrys* paraméterrel az adható meg, hogy legfeljebb hányszor próbálkozzon a nyitással, –ha az esetleg sikertelen volna. Ez abban az esetben hasznos, ha ugyan, az adott port nem foglalt, de az operációs rendszer ezt még nem könyvelte el. 10 másodpercet vár két próbálkozás között. Visszatérési értéke egy szerver socket, illetve, ha nem sikerül socketet nyitni, *RuntimeError* kivételt dob.

#### connect\_client(serverSocket)

Ezzel a függvénnyel egy klienst lehet a szerver socket-re rácsatlakoztatni. Az IP címet és a portszámot kiírja a terminálba, hogy tudni lehessen hova lehet csatlakozni. Ez a függvény blokkoló. Megvárja amíg nem történik egy csatlakozás. Sikeres csatlakozás után kiírja a kliens IP címét és visszatér a kliens socket-jével.

#### close\_socket(clientSocket, serverSocket)

A függvény feladata egy kiépített kapcsolatot lezárni. Mind a kliens, mind a szerver socket-jét bezárja és visszaszolgáltatja az operációs rendszer felé. A bezárt socket IP címét és portszámát kiírja a terminálba.

#### test\_port(portNum)

Feladata igen egyszerű. Meg kell néznie, hogy egy port-ra rá lehet-e csatlakozni vagy sem. Boolean logika szerint tér vissza.

### TCP szerver

A TCP szervernek (*TCP\_server.py*) 2 felelőssége van: TCP kapcsolatból az adatokat kinyerni és azokat formázottan elmenteni. Ehhez szükséges socket nyitása, párhuzamosítás és ezek megfelelő kezelése. Működését az alábbi kód szemlélteti a legjobban:

def main():

thread = threading.Thread(target=thread\_update\_ports)

thread.start()

thread = threading.Thread(target=thread\_release\_bindError\_ports )

thread.start()

while True:

try:

portsSemaphore.acquire()

port = Ports.get\_next\_port()

portsSemaphore.release()

except:

portsSemaphore.release()

continue # Wait until new port frees up

try:

serverSocket = my\_socket.set\_up\_socket(maxRetrys=MAX\_RETRYS, port=port)

clientSocket,serverSocket =my\_socket.connect\_client(serverSocket)

thread = threading.Thread(

target=thread\_socket\_handle, args=(clientSocket,)

)

thread.start()

threadsSemaphore.acquire()

threads.append([serverSocket, clientSocket, thread])

threadsSemaphore.release()

except (ValueError, RuntimeError):

threadsSemaphore.release()

portsSemaphore.acquire()

Ports.set\_port(port=port, used=False, error=True)

portsSemaphore.release()

Elsőre összezavaró lehet a kód, de tárgyalásra kerül minden eleme, és később ugyan ez a kód részletesen magyarázva is lesz. Ekkora már remélhetően szinte evidenssé válik a működése.

#### Adatkezelő segédkönyvtár

A Szerver ugyebár adatokat is kezel, így ezek kezeléséhez érdemes segédkönyvtárat írni. Ez a könyvtár (*data\_handling.py*) felel az adatok validálásáért és megfelelő kinyeréséért.

#### Konstansok

EOM\_CHAR = b"q" # EndOfMessage character

# ^ indicates that the pattern should start matching from the beginning of the input string

# \d+ represents one or more digits

# \. represents a dot

# \s represents one whitespace character (space or tab)

# $ indicates that the pattern should finish matching at the end of the input string

DATA\_PATTERN = pattern = r"^\d+\s\d+\.\d+\s\d+\.\d+$"

A könyvtárnak 2 konstansa van: *EOM\_CHAR* és *DATA\_PATTERN*. Az előbbi arra használatos, hogy ez a karakter jelzi egy logikai adatcsomag végét. Az utóbbi az említett logikai adatcsomagtól elvárt formátumot adja meg.

Példának okáért, a következő sztringek megfelelő formátumúak: „1 2.2 3.3”, „123 456.789 0.1”,

és a következő nem: „1 2.2 3.3” (két space van az 1-es után)

##### is\_data\_valid(data)

Ez a függvény borzasztó egyszerű. Megnézi, hogy a *data* változó formátuma megegyezik-e a *DATA\_PATTERN* által definiált formátummal és ennek megfelelően tér vissza.

if re.match(DATA\_PATTERN, data):

return True

return False

##### extract\_floats(data)

A függvény a *data* változóból kinyer 1 int és 2 float értéket. Ennek indokoltsága a formátum konstansból (*DATA\_PATTERN*) fakad. Megnézi, hogy a formátum helyes-e, és ha nem, *ValueError* kivételt dob. Egyébként pedig visszatér a kinyert adatokkal.

if not is\_data\_valid(data):

raise ValueError

values = data.split()

return int(values[0]), float(values[1]), float(values[2])

##### extract\_data(binaryData)

A függvény egy bináris sztringből nyeri ki az adatokat. Ha a sztringben benne van az *EOM\_CHAR* akkor kinyeri az adatot és kiszámolja, mikor lett kinyerve, majd visszatér a dekódolt adattal, az idővel és a kapott bináris sztringgel. Az időt „YYYY-MM-DD HH:MM:SS” formátumban adja meg.

if EOM\_CHAR in binaryData:

message = binaryData.decode()

message = message.replace(EOM\_CHAR.decode(), "")

# Get current date and time

current\_datetime = datetime.now()

# Format the date and time as a string

formatted\_datetime = current\_datetime.strftime("%Y-%m-%d %H:%M:%S")

return message, formatted\_datetime, binaryData

else:

return None, None, binaryData

#### MyPort

A MyPort osztály (*my\_port.py*) felel a portok megfelelő kezeléséért. Tárolja az állapotaikat és metódusokat biztosít azok frissítéséhez.

##### Belső változók

A belső változója egy lista, aminek egy eleme a következő logikát követi:

[int, bool, bool]=>[portNum, used, bindingError]

A *portNum* változó tárolja hogy melyik port-ról van szó, a *used* tárolja hogy épp használatban van-e az a port és a *bindingError* tárolja hogy volt-e hiba a socket nyitásakor. Fontos megjegyezni, hogy nem érhetőek el változónevek útján a listaelem elemei. Ez a Python egyik tulajdonsága.

Továbbá 3 konstanssal rendelkezik az osztály: *MAX\_PORT\_NUMBERS*, *PORT\_START*, *PORT\_END*

A *MAX\_PORT\_NUMBERS* azt mondja meg, hogy legfeljebb hány port lehet használatban, a *PORT\_START* és a *PORT\_END* pedig egy tartományt definiál, amin belül használhatóak a portok.

##### Konstruktor

A konstruktor három bemeneti paramétert vár: *portStart*, *portMax*, és *portEnd*. Ezek az értékek meghatározzák, hogy milyen porttartományt kezel az osztály. A konstruktor először inicializálja a belső konstansait (*MAX\_PORT\_NUMBERS*, *PORT\_START*, *PORT\_END*), majd ellenőrzi, hogy a megadott porttartomány érvényes-e, azaz a

(portEnd – portStart) > portMax

feltétel teljesül-e. Ha a tartomány érvénytelen, *ValueError* kivételt dob. Ha a porttartomány érvényes, akkor *portStart*-tól kezdve *portMax* számú port-ot úgy inicializál, hogy minden port nem használt (*used*=False) és nincs socket nyitási probléma (*bindingError*=False).

##### get\_next\_port()

Ez a függvény használttá teszi a listában az első nem használt port-ot és visszatér a port számával. Ha minden port használt, *OverflowError* kivételt dob.

##### get\_new\_port(portListElement)

A függvény bemenő paramétere egy teljes *ports* listaelem (Lásd: Belső változók). A funkciója a listában a megadott listaelemet lecserélni egy új, még nem használt port-ra. Ha nem talál ilyet, vagy nincs a megadott elem a listában, *ValueError* kivételt dob. Siker esetén nem-használtként inicializálja az új port-ot és visszatér a számával.

##### set\_port(port, used, error)

Ezzel a függvénnyel módosíthatjuk egy port állapotáról tárolt információkat. Ha nincs ilyen port a listában, hamissal tér vissza, egyébként igazzal.

##### get\_ports\_list()

Ez a függvény egy másolatot szolgáltat vissza a *ports* listáról.

#### Globális változók és konstansok

A Python sajátos tulajdonsága miatt szinte minden változó globális. A Pythonban ezek használata az elsődleges kommunikációs forma, és nem a pointer, mint ahogy C-ben. Ez rossz ötletnek tűnhet, de a Python egy nagyon magas szintű nyelv, így könnyen olvasható és könnyen használhatóak a globális változók különösebb bonyodalmak nélkül.

##### Konstansok

MAX\_PORT = 10 # Maximálisan használható portok száma

PORT\_START = 9000 # Port tartomány kezdeti értéke

PORT\_END = 9999 # Port tartomány végértéke

MAX\_RETRYS = 1 # Hányszor próbálkozzon socket nyitással

TIMEOUT\_SEC = 10 # Mennyi idő után számítson inaktívnak a kliens

A fenti konstansok közül néhányat már bemutattam, egyedül a *TIMEOUT\_SEC*-et még nem. Ez a konstans adja meg azt az időt, ami után ha nem érkezik adat egy socket-en, a kliens inaktívnak lesz feltételezve és bontásra kerül a kapcsolat.

##### Cursor, Connection

Ez a két változó reprezentálja az adatbázist. A script elején jönnek létre és itt csatlakozik a kód az adatbázishoz. Ehhez a két változóhoz tartozik semaphore is.

dbSemaphore = threading.Semaphore(1)

cursor, connection = db\_handling.connect\_to\_db(

host=db\_handling.HOST,

user=db\_handling.USER,

password=db\_handling.PASSWORD,

databse=db\_handling.DATABASE,

)

if not cursor or not connection:

print("Unable to connect to database. Check configuration!")

sys.exit()

A kódban a *db\_handling* a már bemutatott adatbáziskezelő könyvtárat reprezentálja, lásd: Közös, saját adatbáziskezelő segédkönyvtár alfejezet. Hiba esetén a program futása leáll.

##### Ports

A Ports változóval kezelem a port-okat. Ez a MyPort osztály egy példányosítása, és egy van belőle az egész programban. A változóhoz (mint ahogy később majd ez látszik) több szál is hozzá fog férni, így semaphore-al védeni kell.

portsSemaphore = threading.Semaphore(1)

try:

Ports = my\_port.MyPort(portStart=PORT\_START, portEnd=PORT\_END, portMax=MAX\_PORT)

except:

print("Check constants! Incorrect configuration.")

sys.exit()

A *my\_port* a már bemutatott MyPort osztályt tartalmazó file neve, ezen keresztül érhető el az osztály, lásd: MyPort alfejezet. Hiba esetén a program futása leáll.

##### Threads

Ez a változó a majd későbbiekben tárgyalt szálakról hivatott a szükséges információkat tárolni. Ehhez a változóhoz is hozzáfér majd több szál, így védeni kell semaphore-al.

threadsSemaphore = threading.Semaphore(1)

threads = []

Az elképzelés megkönnyítéseként, a következőképpen néz ki egy eleme:

[serverSocket, clientSocket, thread]

Az első két elem már ismert, a harmadik pedig a socket-eket használó szál handle-je.

#### Szálak

A TCP szerver alapvetően szálakon keresztül végezteti el feladatait. Ebben az alfejezetben a szálak működését és feladatait mutatom be. A szálak mind a *TCP\_server.py* file-ban vannak definiálva.

##### thread\_update\_ports()

Ennek a szálnak a feladata a már nem futó szálakat és az ahhoz tartozó információkat eltávolítani a *threads* változóból. A szál nem kap bemenetet és végtelen ciklusban fut, visszatérés nélkül.

while True:

threadsSemaphore.acquire()

for serverSocket, clientSocket, thread in threads:

if not thread.is\_alive():

\_, port = serverSocket.getsockname()

my\_socket.close\_socket(

clientSocket=clientSocket, serverSocket=serverSocket)

portsSemaphore.acquire()

Ports.set\_port(port=port, used=False, error=False)

portsSemaphore.release()

threads.remove([serverSocket, clientSocket, thread])

threadsSemaphore.release()

A logikája viszonylag egyszerű. Megnézi, hogy még fut-e a szál. Ha nem, akkor kinyeri melyik port-ot használta a socket/szál és az ehhez tartozó elemeket törli/frissíti a *threads*-eket és a port-okat nyilvántartó *Ports* változó listájából.

##### thread\_release\_bindError\_ports()

E szál feladata azon port-okat figyelni, amelyeknél socket nyitás során probléma merült fel. A szükségeknek megfelelően frissíti a port-ok információit. A szál nem kap bemenetet és végtelen ciklusban fut, visszatérés nélkül.

while True:

portsSemaphore.acquire()

ports = Ports.get\_ports\_list()

portsSemaphore.release()

for portNum, used, error in ports:

if error:

time.sleep(30) # Wait a little for port to be freed by OS

if not my\_socket.test\_port(portNum=portNum): # still in use

portsSemaphore.acquire()

Ports.get\_new\_port([portNum, used, error])

portsSemaphore.release()

continue

portsSemaphore.acquire()

Ports.set\_port(port=portNum, used=False, error=False)

portsSemaphore.release()

A szál először másolatot készít a port-ok listájáról. Erre azért van szükség, mert esetekben 30 másodperc többszöröséig is foglalt lehet a *Ports* erőforrás és fölöslegesen váratná a többi szálat, amelyek erre az erőforrásra várnak. A hosszú erőforrás foglalás helyett csak a módosítás eszközölésének idejéig foglalja az erőforrást. A szál egyesével végignézi, melyik port-nál történt socketnyitási hiba. Ahol hiba történt, ott vár 30 másodpercet, –hátha elengedi az operációs rendszer és megoldja a problémát. Ha ez nem történt meg, azt a port-ot törli a listából és kér helyette újat. Ha az operációs rendszer megoldotta a problémát, törli a hibajelzést.

##### thread\_socket\_handle(clientSocket)

Ez a szál kezeli ténylegesen a TCP kapcsolaton keresztül érkező adatokat. Felel a TCP kapcsolaton közölt adatok kinyeréséért, azok helyességéért és azok elmentéséért. A szál folyamatát a 13. ábra szemlélteti.



13. ábra

Az adat először bináris formában érkezik TCP kapcsolaton keresztül. A szál megvárja amíg egy teljes, logikailag egybetartozó adatcsomag megérkezik, azaz *EOM\_CHAR*-ral lezáródik az adatsorozat. Ha *TIMEOUT\_SEC* ideig nem érkezik semmi a socket-en, visszatér a szál. A folyamat bármelyik pontján érkezhet egy FIN flag, amely a kapcsolat bontását jelzi a kliens részéről, ilyenkor is visszatér. Ha megjött a teljes adatcsomag, leellenőrzi, hogy megfelel-e az elvárt formátumnak, ami a *data\_handling.py*-ban van megadva. Helyes formátum esetén az adatbázisba írja az adatokat, egyébként meg figyelmen kívül hagyja és vár egy új adatcsomagra.

#### Algoritmus

A *thread\_socket\_handle* szál önmagában teljes mértékben képes kezelni egy socket-et és az azon keresztül érkező adatokat. Ez azt jelenti, hogy minden kapcsolathoz rendelni kell egy ilyen szálat. Az ehhez szükséges előkészületeket és az szálak indítását végzi a *main* függvény (nem keverendő a későbbiekben a *main.py* file-al), az alábbi kód. Ugyan a Python nem igényel main függvény definiálást, de C-n keresztül tanultam meg elsőnek programozni, így szinte egyértelmű volt számomra, hogy ezt így kell csinálni. A *main* függvényt a script hívja meg.

thread = threading.Thread(target=thread\_update\_ports)

thread.start()

thread = threading.Thread(target=thread\_release\_bindError\_ports )

thread.start()

while True:

try:

portsSemaphore.acquire()

port = Ports.get\_next\_port()

portsSemaphore.release()

except:

portsSemaphore.release()

continue # Wait until new port frees up

try:

serverSocket = my\_socket.set\_up\_socket(maxRetrys=MAX\_RETRYS, port=port)

clientSocket = my\_socket.connect\_client(serverSocket)

thread = threading.Thread(target=thread\_socket\_handle, args=(clientSocket,))

thread.start()

threadsSemaphore.acquire()

threads.append([serverSocket, clientSocket, thread])

threadsSemaphore.release()

except (ValueError, RuntimeError):

threadsSemaphore.release()

portsSemaphore.acquire()

Ports.set\_port(port=port, used=False, error=True)

portsSemaphore.release()

Mivel a cél a TCP kapcsolatok szálakon keresztüli kezelése, ezért e körül forog az algoritmus. A portok kezelését a *Ports* változó végzi. A megfelelő kezeléshez szükségesek a *thread\_update\_ports* és *thread\_release\_bindError\_ports* szálak is. Ezek automatikusan elvégzik a felmerülő hibák kezelését, így már csak az marad, hogy minden lefoglalt port-hoz új *thread\_socket\_handle* indítódjon.

Először elkéri az algoritmus az első szabad port-ot. Ha nincs ilyen, addig próbálkozik amíg nem lesz.

Miután kapott szabad port-ot, kér az operációs rendszertől egy socket-et és megvárja amíg ehhez a socket-hez nem kapcsolódik egy kliens. Ha kapcsolódott a kliens és kiépült a kapcsolat, már indítható is hozzá egy *thread\_socket\_handle* szál. Ha az ebben a bekezdésben leírtakkor bármilyen hiba fellép akkor kötési hibát rendel ahhoz a port-hoz az algoritmus.

A kötési hibákat ugyebár a *thread\_release\_bindError\_ports* szál kezeli, a már nem használt port-okt pedig a *thread\_update\_ports* kezeli*.* Mivel a *thread\_socket\_handle* a kapcsolat bontását visszatéréssel jelzi, így már indokolt és érthető a *thread\_update\_ports* szükségessége/viselkedése/algoritmusa.

### Backend

A Backend-nek két feladata van: egy weboldal file-jait kiszolgáltatni, valamint annak a weboldalnak API-ként szolgálni. Mindkét funkciót a *Flask* könyvtár segítségével megvalósítottam meg. A Backend egy külön modulként van definiálva. A *Flask* használatához egy úgynevezett *app*-ra van szükség, melyet a következőképpen kell létrehozni:

ROOT\_DIR = os.path.dirname(os.path.abspath(\_\_file\_\_))

ROUTE\_DIR = os.path.join(ROOT\_DIR, "../../Frontend/angelos/build/")

app = Flask(\_\_name\_\_, static\_folder=ROUTE\_DIR)

A *ROOT\_DIR* változó megadja annak a könyvtárnak az elérési útvonalát, amiben a file van. A ROUTE\_DIR pedig ehhez képest a következő útvonalat adja meg:

Server/

|-- \_\_ init\_\_.py

|-- main.py

|-- backend/

|-- web\_backend.py // innen indul

Frontend/

|-- \_\_ init\_\_.py

|-- main.py

|-- angelos/

|-- build/

|-- files // az elért file-ok

Ez azt jelenti, hogy a *build* könyvtárból a *files* elérési útvonalán lévő file-okat éri el közvetlen az *app*.

#### Globális változók

A Backend API része használja az adatbázist, úgyhogy itt is szükség van a megfelelő változókra.

cursor, connection = DB.connect\_to\_db(

host=DB.HOST, user=DB.USER, databse=DB.DATABASE, password=DB.PASSWORD)

#### Webszerver

A lenti kód valósítja meg a Szerver Webszerver részét.

@app.route("/manifest.json")

def serve\_manifest():

return app.send\_static\_file("manifest.json")

@app.route("/static/<path:filename>")

def serve\_static(filename):

return send\_from\_directory(ROUTE\_DIR + "/static", filename)

@app.route("/", defaults={"path": ""})

@app.route("/<path:path>")

def catch\_all(path):

return app.send\_static\_file("index.html")

Az *@app.route* itt azt mutatja, hogy milyen URL-ű HTTP kérés esetén fog az alatta lévő függvény lefutni. Ennek a kiértékelése fentről lefelé történik, így fontos a sorrend.

##### Serve\_manifest

Ez a függvény a *ROUTE\_DIR* könyvtárból küldi ki a *manifest.json* file-t.

##### Serve\_static

A *serve\_static* a *static* könyvtár file-jait képes visszaküldeni. A *<path:filename>* azt éri el, hogy ha az URL egészen a *static* alkönyvtárig egyezik, akkor a file nevét paraméterként megadja a függvénynek. Így a megfelelő URL-lel a *static* könyvtár minden file-ja kiszolgáltatható.

##### Catch\_all

Az *catch\_all* annyit tesz, hogy minden URL esetén az *index.html* file-t küldi ki.

##### Algoritmus

Az algoritmusa egész egyszerű. Fentről lefelé az első lehetőségnél teljesíti a HTTP kérést. Annyi a csavar benne, hogy előbb minden konkrét URL-t kiszolgál, majd a *catch\_all*-al minden más URL esetén az *index.html*-t küldi vissza.

#### API

Az API a következő endpoint-on van:

@app.route("/api/data", methods=["GET"])

, és két paramétert fogad el: *id* és *unique\_ids.*

##### Id

Az *id* paraméterrel ellátott endpoint az adatbázisból egy adott id-hoz tartozó adatokat adja vissza, formázottan. Az alábbi kód felel ezért:

db\_semaphore.acquire()

data = DB.send\_query(

cursor=cursor,

connection=connection,

query=f"SELECT humidity, temperature, time\_of\_measurement FROM measurements WHERE sensor\_id = {id};",)

db\_semaphore.release()

db\_semaphore.acquire()

dataMin = DB.send\_query(

cursor=cursor,

connection=connection,

query=f"SELECT MIN(humidity) AS min\_humidity, MIN(temperature) AS min\_temperature FROM measurements WHERE sensor\_id = {id};",)

db\_semaphore.release()

humidityMin, temperatureMin = dataMin[0]

formattedTempData = [

{"date": time.strftime("%Y-%m-%d %H:%M:%S"), "value": temperature}

for \_, temperature, time in data]

formattedHumidData = [

{"date": time.strftime("%Y-%m-%d %H:%M:%S"), "value": humidity}

for humidity, \_, time in data]

return jsonify(

{

"temp": formattedTempData,

"humid": formattedHumidData,

"min\_humidity": humidityMin,

"min\_temperature": temperatureMin,})

Ebben az esetben az API 3 oszlopot kér le, a *measurements* táblából, az adatbázisból, minden olyan sor esetén, ahol az *id* megegyezik a *sensor\_id*-val. Ezek az oszlopok rendre: *humidity*, *temperature* és *time\_of\_measurement.* Ezután párokba rendezi az adatokat: *hőmérséklet* és *mérési idő*, valamint *páratartalom* és *mérési idő*. A mérési időt mindkét esetben „YYYY-MM-DD HH:MM:SS” formátumúvá konvertálja át. Az adatokat a *date* és *value* címkékkel is ellátja. Végső soron 1-1 *temp* és *humid* címkével ellátva az adatokat elküldi a kérő részére.

A második lekérdezés esetében a lekérdezett adatok minimumait kéri le. Ezeket a *min\_humidity* és a *min\_temperature* címkékkel ellátva küldi vissza. Adatbázis használatok semaphore-al védve vannak.

Fontos megjegyezni, hogy GET paraméterként várja az *id*  értéket, így egy helyes API kérés URL-je például: „/api/data?id=1”.

##### Unique\_ids

Ezt az API endpoint-ot a következő kód szolgálja ki:

db\_semaphore.acquire()

data = DB.send\_query(

cursor=cursor,

connection=connection,

query=f"SELECT DISTINCT sensor\_id FROM measurements;",)

db\_semaphore.release()

data = [id for id, in data]

return jsonify({"sensor\_id\_array": data})

Feladata egyszerű. Vissza kell küldenie a *measurements* tábla egyedi *sensor\_id*-jai listáját. Ezt a *sensor\_id\_array* címkével ellátva küldi vissza.

### Algoritmus

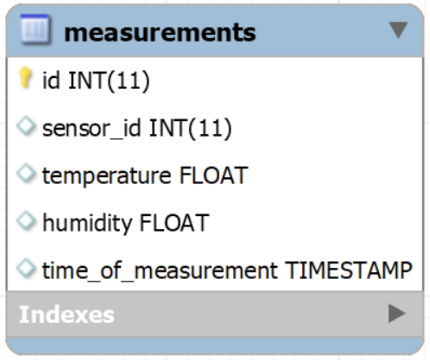
A Szerver a TCP szervert és a Backend-et egyszerre indítja el 1-1 alfolyamatként, így párhuzamosan futnak. Mindkét algoritmust a gép alhálózatbeli IP címén indítja el a socket-kezelő könyvtár *get\_local\_ip()* függvényének segítségével. Egy beéptett változón keresztül lehet választani, hogy debug vagy éles szerver induljon el. Mindkét esetben az 5000-es port-on próbálkoznak. A két folyamat bármikor törölhető a *crtl+c* billentyűkombinációval. Az algoritmust a *main.py* valósítja meg.

## Adatbázis

Az Adatbázis szerepe a project-ben pusztán adatok tárolása. Ez a komponens a legegyszerűbb az összes közül.

### Tábla

Az adatbázis egy táblából áll, amelyet a 14. ábra szemléltet.



14. ábra

Az 5 elemből csak az *id* és a *time\_of\_measurement* szorulhat némi magyarázatra.

#### Id

Ez a tábla elsődleges kulcsa, ami azt jelenti, hogy minden adatsornak egyedi azonosítója lesz. Ez int típusú és automatikusan növekszik minden új rekorddal.

#### Time\_of\_measurement

Ez a mező *timestamp* típusú, ami azt jelenti, hogy „YYYY-MM-DD HH:MM:SS” formátumban tárolja a dátumot.

### Csatlakozás

Csatlakozni kizárólag localhost-on lehet ra. Az adatbázison egy felhasználó van beállítva „pythonUser” felhasználónévvel és „pythonUser” jelszóval.

## Weblap

A weboldal adatok vizuális megjelenítéséért felel. Ehhez a React JavaScript könyvtárat használtam fel. A React komponens szemléletű logikája segítségemre volt a weboldal létrehozásában. Az adatokat chart-ok segítségével jelenítem meg, amelyhez a [DevExtreme Zooming and Panning API](https://js.devexpress.com/React/Demos/WidgetsGallery/Demo/Charts/ZoomingAndScrollingAPI/MaterialBlueLight/)-ját[42] használtam. A komponensfa struktúráját, az átadott paramétereket és az API hívásokat a 15. ábra szemlélteti.



15. ábra

### Könyvtárak

Számos kiegészítő könyvtárat használtam fel a fejlesztés során. Ezek mind valamely fontos funkció megvalósításában voltak a segítségemre. Ebben a fejezetben csak egy átfogó képet szeretnék alkotni a használatukról és a hasznukról. A projectben felvett tényleges szerepeikről és felhasználásukról később esik szó. Hozzá szeretném tenni, hogy a következőkben említett könyvtárak funkciója messze meghaladja az általam említetteket! Ezek nagyon sokrétű, terjedelmes könyvtárak, amelyeknek én csak a töredékét használtam fel.

#### React Router Dom

A React Router Dom egy olyan könyvtár, amely a [React Router](https://reactrouter.com/en/6.20.1)[43] könyvtárra épít. A projectben a 6.20.1-es verziója került felhasználásra. A könyvtár a különböző navigációs formák (link, átirányítás stb.) implementálásában segít.

**Browser router**

Ez a komponens teszi elérhetővé a navigálást a projecten belül. Az egész projectben való használathoz a teljes kódot körbe kell venni a következő komponenssel:

<BrowserRouter>

// A project többi komponense

</BrowserRouter>

**Routes, Route és useParam**

Ez a két komponens azt teszi lehetővé, hogy az egyoldalas React projectben URL függően más és más komponenseket lehessen megjeleníteni. A *<Routes>* komponens magába foglalja a különböző URL-eket, *<Route>*-okat. Ennek a kiértékelése felülről lefelé történik, mindig az első *path* egyezés fog kiértékelődni. Ekkor az *element* kerül renderelésre. Az URL-eken keresztül lehet paramétereket is átadni (/:param). A zárójelezett esetben a paraméter a *param* néven és a *useParam* függvénnyel kérhető el.

<Routes>

<Route path="/path/:param" element={<Element />} />

</Routes>

const { param } = useParams();

**Link és NavLink**

A Link és a NavLink komponensek nagyon hasonlóak. Mind a kettő egy HTML <a> (link) taget helyettesít. Úgy irányítanak át, hogy azt a *Routes* komponenssel át lehessen irányítani az oldalon belül. A *NavLink* annyi pluszt tesz hozzá, hogy automatikusan ki tudja emelni (pl. kiszínezi) azt a linket (pl. egy leugró listában) ami épp be van töltve.

#### Axios

Az [Axios](https://www.npmjs.com/package/axios/v/1.6.2)[44] (v. 1.6.2.) egy HTTP kliens, amellyel API hívást lehet megvalósítani. A *get* metódusával megadható, hogy milyen URL-ről kérjen adatot. Ez az *await* kulcsszóval párhuzamosítható, így nem hátráltatja az egyéb folyamatokat.

#### React query

A [React query](https://www.npmjs.com/package/react-query/v/3.39.3)[45] (v. 3.39.3) egy olyan könyvtár, amelynek segítségével API hívást és cach-elést lehet kezelni.

**useQuery**

A *useQuery*-vel lehet egyben az API hívást és a cach-elést is megoldani. Az elmentett adatokat egy egyedi *queryKey*-vel menti el, így egyértelműen azonosítható lesz. Meg lehet adni azt is, hogy mennyi idő után frissítse az elmentett adatot.

### Segédfüggvény

A weboldalhoz szerencsére csak 1 segédfüggvényre van szükség és az is meglehetősen egyszerű. A függvény kódja a következő:

export function transformRangeSelectorData(data, dataMin) {

return data.map((item) => ({

date: item.date,

value: item.value - dataMin,

}));

}

Paraméterként egy osztálytömböt és egy számértéket kap. Az osztálytömb elemei között a *date* és a *value* címkékre számít. A függvény minden *value* értékből kivonja a *dataMin* számértéket, majd visszatér az új tömbbel.

### Komponensek

A komponensek bemutatását a 15. ábra szerint lentről felfelé teszem meg. Azért ezt az elvet követem, mert a feljebb álló komponensek tipikusan valamilyen kiegészítő-, illetve más komponensekkel való összefogó/összekötő funkciókat látnak el.

#### MyChart

A MyChart osztály egy amolyan wrapper (burkoló) komponens a *DevExtreme Zooming and Panning API*-jához. Az API itt azt jelenti, hogy a *DevExtreme* által implementált *Chart*-hoz kiegészítő funkciókat készítettek, mint a nagyítás és a mozgatás. A [42]-es irodalomjegyzék linkjén meg van adva egy példakód, ezt használtam fel némi módosításokkal. Az API-n belül az ábrázolás folyamatának teljessége kimerítően bonyolult és számomra sem teljesen ismert, így csak a legfontosabb tudnivalókat mutatom majd be róla.

**Kapott paraméterek**

A MyChart komponens 2 paramétert kap: a megjelenítendő adatot (*data)* és a megjelenítendő adat minimumát (*dataMin)*. Az adat itt egy osztálytömböt jelent, amely osztályoknak 2 attribútuma van: *value* és *date*. A *dataMin* a *value* adatok közül a legkisebb.

**Belső változók**

A belső változók mindegyike a konstruktorban van inicializálva, külön definiálásra a React-ben nincs szükség.

constructor(props) {

super(props);

this.state = {

visualRange: { startValue: undefined, endValue: undefined },

rangeSelectorData: transformRangeSelectorData(props.data, props.dataMin),

};

this.chartRef = React.createRef();

this.falseVisibility = { visible: false };

}

A *visualRange* a megjelenítendő adattartományt tárolja. A kezdeti értéke nincs meghatározva.

A *rangeSelectorData* a kapott *data* változó *dataMin*-nel eltolt verziója.

Az *chartRef* egy olyan változó, amely segítségével interakcióba lehet lépni a DOM elemeivel. Ezt majd valamihez hozzá kell rendelni hogy használni lehessen.

A *falseVisibility* egy egyszerű osztály, amelynek a *visible* attribútuma hamis.

**ComponentDidUpdate(prevProps)**

A ComponentDidUpdate egy úgynevezett lifecycle hook. Ez azt takarja, hogy ha valami a weblap életében (innen a lifecycle elnevezés) megváltozik, akkor a változásnak megfelelő helyen tud kapaszkodni (innen a hook elnevezés) a weboldal.

Jelen esetben két funkciója van:

Elsőként, a *chart*-ot le kell frissítenie. Azt tapasztaltam, hogy esetekben nem fért ki minden információ az y tengelyre és egy frissítés hatására már viszont hiánytalanul kifért minden. Ezt a következő kódrészlet teszi meg:

const { current } = this.chartRef;

if (current && current.instance) {

current.instance.render({

force: true, // forces redrawing

animate: false, // redraws the UI component without animation

});

}

Megnézi, hogy létezik-e jelenleg a *chart* és ha igen, akkor kirajzoltatja újra.

Második funkciója az, hogy ha új adatokat kap a komponens, azokat frissíteni kell. Ezt a következőképpen teszi:

if (prevProps.data !== this.props.data) {

this.setState({

rangeSelectorData: transformRangeSelectorData(

this.props.data,

this.props.dataMin

),

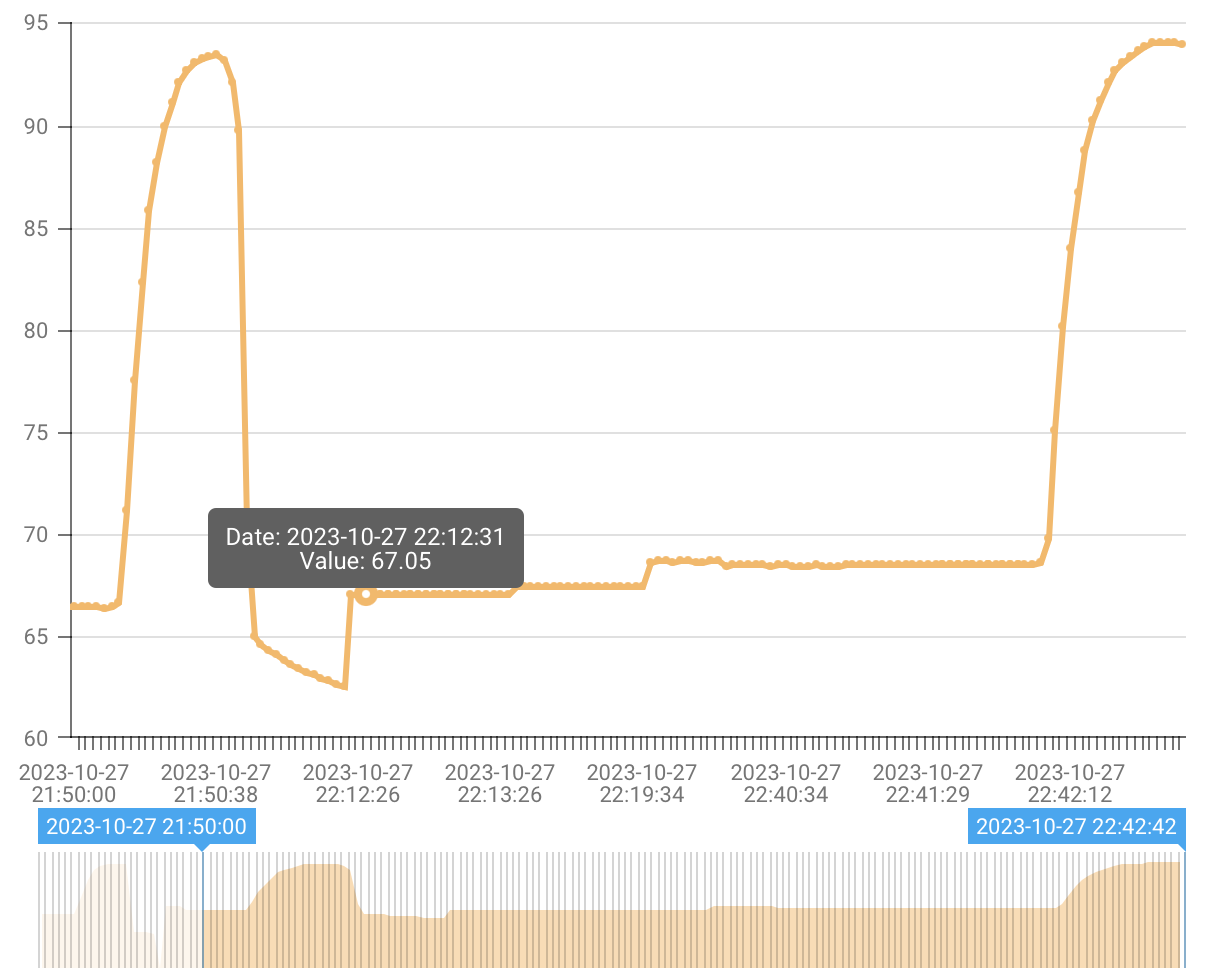
});

}

Ha az eddigi adatok különböznek az új adatoktól, akkor frissíteni kell az egyetlen frissíteni való változót, a *rangeSelectorData*-t. Ezt meg is teszi.

**DevExpress Chart**

A chart funkcióit a 16. ábrán keresztül szeretném bemutatni, amely valós, a szakdolgozatban tárgyalt Adatgyűjtő által mért adatokat jelenít meg a többi komponens segítségével (Szerver és adatbázis). A chart két részre bomlik: adatablak és nagyítósáv. Az adatablak a 16. ábra fenti része a sárga vonalábrával, a nagyítósáv pedig az alsó, két kék címkét tartalmazó téglalap.



16. ábra

**Adatablak**

Az adatablakért az alábbi kód felel:

<Chart

ref={this.chartRef}

id="zoomedChart"

palette="Harmony Light"

dataSource={data}>

<Series argumentField="date" valueField="value" />

<ArgumentAxis visualRange={this.state.visualRange}></ArgumentAxis>

<Legend visible={false} />

<CommonSeriesSettings>

<Point size={4} />

</CommonSeriesSettings>

<ValueAxis>

<Label format="decimal" />

</ValueAxis>

<Tooltip

enabled={true}

customizeTooltip={(pointInfo) => {

return {

text: `Date: ${pointInfo.argumentText}\nValue ${pointInfo.valueText}`,

};

}}

/>

</Chart>

Az adatablakról legfontosabb információ az, hogy hogyan lehet megadni neki az adatokat. Az adatokat a *Chart* *dataSource* tulajdonságaként lehet megadni. Itt egy osztálytömböt vár. A *Series* komponensben lehet megadni, hogy milyen attribútumnéven keresse az *argumentAxis* (x tengely) és a *valueAxis* (y tengely) adatait (*dataSource)*. Jelen esetben a *date* és a *value* címkenevek alatt keresi.

Egyedi Tooltip-van (kurzort az adatot reprezentáló pontra víve Tooltip jelenik meg) beállítva, melyet a következő függvény tesz lehetővé:

(pointInfo) => {

return {

text: `Date: ${pointInfo.argumentText}\nValue: ${pointInfo.valueText}`,

};

}

Ez a 16. ábrán látható kis téglalapot eredményezi.

A nagyítás beállításához a megjelenítendő tartományt (*visualRange*) kell módosítani. Ezt majd a nagyítósáv végzi el.

**Nagyítósáv**

A nagyítósáv kódja a következő:

<RangeSelector

dataSource={rangeSelectorData}

onValueChanged={(e) => {

this.setState({ visualRange: e.value });

}}>

<Size height={80} />

<Margin left={10} />

<Scale

startValue={rangeSelectorData[0].date}

endValue={rangeSelectorData[rangeSelectorData.length - 1].date}

label={this.falseVisibility}

/>

<ChartOptions palette="Harmony Light">

<Behavior valueChangeMode="onHandleMove" />

<Legend visible={false} />

<Series argumentField="date" valueField="value" />

</ChartOptions>

</RangeSelector>

A nagyítósáv segítségével lehet nagyítani az adatablakon. Ezt a nagyítósávban a két kék címke mozgatásával lehet megtenni. A nagyítósáv megjeleníti az adatablak is tartalmát, csak kicsiben. A kezdetben az volt a tapasztalatom, hogy ugyan megjeleníti az adatot, a változások nem vehetőek ki rajta kellően. Ezért használom a *rangeSelectorData*-t, mert így (hogy el van tolva), sokkal látványosabb lesz a nagyítósávban megjelenített adatban a változások. A *DevExpress* chart-ját nem sikerült úgy módosítanom, hogy ezt a hatást el tudjam érni.

Az *onValueChanged* paraméternek egy függvényt kell megadni, amely kezeli az esetleges változtatásokat a megjelenítési tartományban. A függvényt helyben implementáltam, ami frissíti a *visualRange*-t.

A *Scale* komponenssel meg tudtam adni, hogy hol legyen a nagyítósáv két kék címkéje a kezdetben. A *startValue* és az *endValue* felel ezekért. A *rangeSelectorData* elejére illetve végére állítottam be őket.

A *Series* komponenssel itt is azt kell beállítani, amit az adatablaknál, azaz milyen címke alatt vannak a tengelyek adatai.

**Render()**

A *render* felelős a virtuális DOM elemeinek létrehozásáért és visszaadásáért, amelyek a komponens által megjelenített UI-t reprezentálják, azaz a 16. ábrán látottakat.

render() {

const { data } = this.props;

const { rangeSelectorData } = this.state;

return (

<div className="container">

// DevExpress chart

</div>

);

}

}

A *render* kinyeri az adatablak és a nagyítósáv által megjelenítendő adatokat (*data* és *rangeSelectorData*), majd a chart-ot egy *div*-be burkolva visszaadja. A *container* osztálynév egy kis margót ad a benne lévő komponenseknek, így kicsit szebbé téve az UI megjelenését.

#### Charts

A Charts a MyChart-tal ellentétben nem egy osztály, hanem egy SFC (Stateless Functional Component, állapotmentes funkcionális komponens). Külön állapottal nem rendelkezik, mint ahogy a MyChart, így kicsit máshogy is működik. Nincsnek lifecycle hook-jai, és a *render* metódus itt csak egy sima *return*. Egy SFC úgy viselkedik, mint egy függvény.

Ezen SFC egy táblázatban 1-1 MyChart komponenst jelenít meg. Ő szolgáltatja az adatot is a MyChart-oknak. Az adatokat elcacheli, így lekérdezésükre csak egyszer van szükség. Ez sok erőforrást spórol meg. Segítségképpen, a 17. ábrán látható a Charts komponens, ez kerül ebben az alfejezetben bemutatásra.

A screenshot of a graph

Description automatically generated

17. ábra

**Kapott paraméterek**

A Charts 1 paramétert kap közvetlen, az őt hívó komponenstől: *SERVER\_API\_ADDR*. Ez a paraméter megadja az API elérési útját (nem az endpoint-ét).

További 1 paramétert kap, méghozzá URL-en keresztül, egy *id*-t, melyet majd az API endpoint hívásakor használ fel.

const { id } = useParams();

**API hívás**

Mivel tulajdonképpen ez a komponens felel az adatok megjelenítéséért, így neki kell el lekérnie az adatokat. A következő kódsorok valósítják ezt meg:

// Külön függvénydefiníció az SFC-n kívül

const fetchData = async (id, ADDR) => {

const response = await axios.get(`${ADDR}?id=${id}`);

return response.data;

};

const queryKey = ["data", id];

const { data, error, isLoading } = useQuery(

queryKey,

() => fetchData(id, SERVER\_API\_ADDR)

);

if (isLoading) {

return <p>Loading data...</p>;

}

if (error) {

return <p>Error: {error.message}</p>;

}

const { humid, temp, min\_humidity, min\_temperature } = data;

Az API lekérdezést és a cachelést a *useQuery*-vel végzi. Ehhez egy egyedi kulcsot állít elő. A kulcs felhasználásával elvégzi az endpoint hívást, amely: *?id=x*, ahol x a lekérdezendő adatok *id*-ja. Az API hívás után a megfelelő státuszinformációkat megjeleníti weboldalon, majd, ha már megérkezett az adatcsomag, kinyeri a megfelelő címkéjű adatokat.

**Return**

Az SFC-k esetében pusztán a *return* felel az UI megjelenítéséért, mely az alábbi:

return (

<React.Fragment>

<h1>showing {id}</h1>

<table className="table">

<thead>

<tr>

<th scope="col">Humidity</th>

<th scope="col">Temperature</th>

</tr>

</thead>

<tbody>

<tr>

<td>

<MyChart data={humid} dataMin={min\_humidity} />

</td>

<td>

<MyChart data={temp} dataMin={min\_temperature} />

</td>

</tr>

</tbody>

</table>

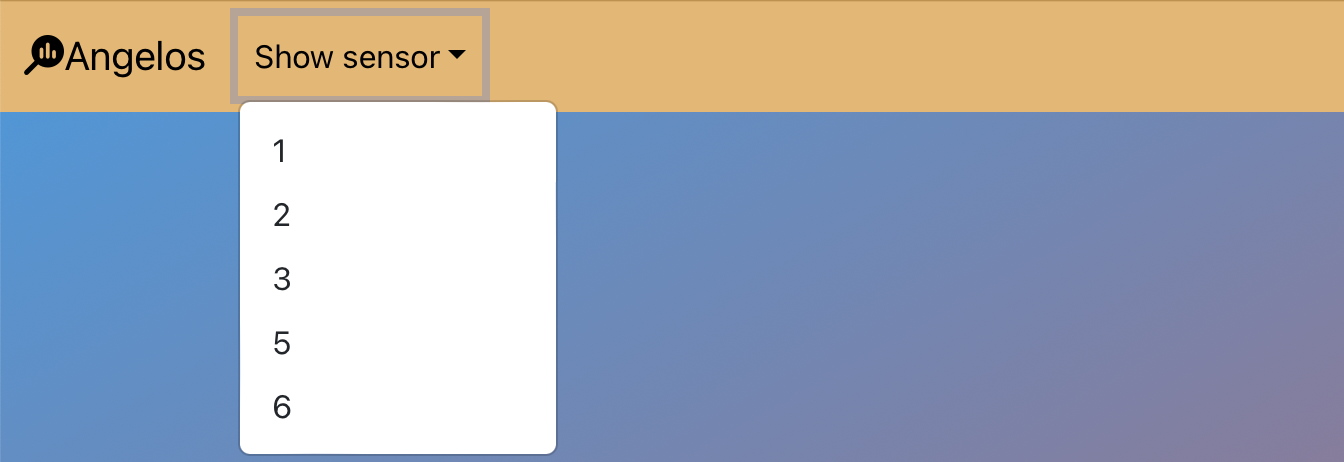
</React.Fragment>

);

A Charts egy [Bootstrap táblázattal](https://getbootstrap.com/docs/4.0/content/tables/#examples)[45] tér vissza, amely bal oldalán páratartalom adatok, jobb oldalán hőmérséklet adatok jelennek meg. A táblázat felett megjelenik, hogy melyik *id*-hoz tartozó adatokat lehet látni.

#### NavBar

A NavBar is egy SFC, amely egy navigációs sávot valósít meg és használja a Bootstrap-et és a FontAwesome-ot is. A NavBar a 18. ábrán a fenti aranybarna csík a leugró listával (ez nincs mindig ott). A kék háttér nem része.



18. ábra

**Kapott paraméterek**

A sáv egy tömböt kap, amelyben a leugró listában megjelenítendő számokat várja.

**UI**

A navigációs sáv egy [Bootstrap Navbar](https://getbootstrap.com/docs/4.0/components/navbar/#nav)-t[46] vesz alapul.

**Esztétika**

Az UI háttérszínét és elrendezését a következő sorok adják:

<nav className="navbar navbar-expand navbar-light"

style={{ backgroundColor: "rgb(236, 181, 107)" }}

Meg vannak adva a Bootstap osztályok (amelyeket a [46]-os irodalomjegyzék részletez) , és a háttérszín. A *navbar-light* a betűtszínt feketére állítja.

**Brand ikon**

Az Angelos feliratot és a [nagyító ikon](https://fontawesome.com/icons/magnifying-glass-chart?f=classic&s=solid&rt=flip-horizontal)-t[47] a következők valósítják meg:

<Link className="navbar-brand" to="/">

<FontAwesomeIcon icon={faMagnifyingGlassChart} flip="horizontal" />

Angelos

</Link>

Az *Angelos* szöveg és az ikon is kattinthatóak. A kattintás a felhasználót a „*/*” URL-re irányítja.

**Leugró lista**

A kattintható leugró lista kódja az alábbi:

<div className="collapse navbar-collapse">

<ul className="navbar-nav">

<li className="nav-item dropdown">

<a

className="nav-link dropdown-toggle"

href="#"

id="navbarDropdown"

role="button"

data-bs-toggle="dropdown"

aria-expanded="false">

Show sensor

</a>

<ul className="dropdown-menu" aria-labelledby="navbarDropdown">

{ids.map((id) => (

<NavLink

className="dropdown-item"

to={`/sensor/${id}`}

key={id}>

{id}

</NavLink>

))}

</ul>

</li>

</ul>

</div>

A *Show sensor* szöveg kattintható, mely hatására kilistázza a kapott tömb minden elemét. Minden elemhez egy kattintható linket társít, amely a *„/sensor/x”* URL-re irányítja a felhasználót, ahol *x* a kiválasztott elem száma.

**Return**

A NavBar komponens a következőképpen tér vissza az UI-kkal:

return (

<nav

className="navbar navbar-expand navbar-light"

style={{ backgroundColor: "#a7826ca2" }}>

<div className="container-fluid">

// Brand ikon

// Leguró lista

</div>

</nav>

);

Az említett UI-k egy Bootstrap *container-fluid*-ban lesznek kirajzolva, ami azt teszi, hogy dinamikusabbá válik tőle a benne lévő tartalom, nem fog lecsúszni a képernyőről képernyőméretváltoztatás esetén.

#### TextRotator

A TextRotator is SFC. Feladata egy üdvözlőszöveget kirajzolni. Mivel ez egy elég egyszerű komponens, nem szükséges részegységekre bontva tárgyalni. A 19. ábra mutatja az UI-t. A képen a *Hello User!* felirat a térben forog, balról jobbra.



19. ábra

A kódja:

const TextRotator = ({ username }) => {

return (

<div className="text-rotator">

<div className="text-content">

<h1 className="username">Hello, {username}!</h1>

<h2 className="welcomeMessage">Welcome to Angelos!</h2>

</div>

</div>

);

};

export default TextRotator;

A *return* logikája elég egyszerű és egyértelmű. A paraméteként kapott sztringet megjeleníti egy üdvözlőszöveg részeként. A kinézet megvalósítására több CSS osztályt hoztam létre.

**Text-rotator**

.text-rotator {

position: fixed;

width: 100%;

height: 100%;

display: flex;

justify-content: center;

align-items: center;

perspective: 1200px; /\* Set the perspective for 3D effect \*/

background: linear-gradient(

to bottom right,

#3498db,

#e74c3c

); /\* Apply gradient background \*/

background-size: cover; /\* Ensure the background covers the entire container \*/

overflow: hidden;

}

A *position*, *width*, *height*, *display*, *justify-content* és *align-items* együttesen felel a szöveg középre kerüléséért. A *perspective* megadja, hogy milyen perspektívából látszon a megjelenített tartalom. A *background* egy lineáris átmenetet ad háttérnek, két szín között, a bal felső sarokból, a jobb alsó sarokig. A *background-size* biztosítja, hogy a képernyőn minden számára biztosított helyet kitöltsön. Az *overflow* azt teszi, hogy ha a tartalom kicsordulna az ő konténeréből, akkor ne folyjon át másik konténerbe.

**Text-content**

.text-content {

transform-style: preserve-3d;

text-align: center; /\* Center the text horizontally \*/

}

A *transform-style* meghatározza, hogy a gyermek elemek tartsák meg az őket érintő 3D-s transzformációkat. A *text-allign* vízszintesen középre igazítja a szöveget.

**Username**

.username {

color: rgb(236, 181, 107);

font-size: 4rem;

font-weight: bold;

text-transform: uppercase;

animation: rotate 6s linear infinite;

display: inline-block;

}

A *color*, *font-size*, *font-weight és text-transform* mind a betűk megjelenési stílusáért felelnek. Az *animation* egy 6 másodperces, lineáris, végtelenségig tartó animációt definiál. A *display* itt a blokk típusát definiálja.

**WelcomeMessage**

.welcomeMessage {

color: rgb(236, 181, 107);

font-size: 4rem;

font-weight: bold;

text-transform: uppercase;

font-family: "Times New Roman", Times, serif;

}

Ennél a CSS osztálynál minden tag a betűk stílusáért felel. A *username* osztálytól csak annyiban különbözik, hogy más a betűtípus és nincs transzformáció definiálva.

**Transzformáció**

@keyframes rotate {

0% {

transform: rotateX(0deg) rotateY(0deg);

}

100% {

transform: rotateX(0deg) rotateY(360deg);

}

}

A transzformáció egy balról jobbra történő 360 fokos fordulat. A CSS 3D térben gondolkodik, de Descartes koordinátarendszerben ez a transzformáció az x tengelyen történik.

#### App

Az App komponens a komponensfa gyökere. Ez a komponens felel a megfelelő oldalátirányításokért, valamint egy API hívásért is. Lévén, hogy a többi komponens felett áll, így nem kap paramétert. Ez a komponens osztályként van definiálva.

**Belső változók**

Az osztály két belső változóval rendelkezik:

state = {

sensor\_id\_array: [],

SERVER\_API\_ADDR: "http://152.66.158.66:5000/api/data",

};

A s*ensor\_id\_array* üresen van inicializálva és itt fogja eltárolni a majdani API hívás adatait. A második változó egy elérési út az API-hoz.

**ComponentDidMount**

A componentDidMount-ban történik az API hívás, de előtte még frissíti az API elérési útját, ha szükséges. A szükséges itt azt jelenti, hogy ha az előre megadott elérési út IP címe „*localhost”*, akkor kinyeri a weboldal URL-jéből az IP címet és a portszámot, majd módosítja a *SERVER\_API\_ADDR* változót. Erre pusztán azért van szükség, mert az npm debug szervere mindig localhost-on nyílik meg, így ilyenkor használható a beépített elérési út, viszont éles tesztelés közben már sokkal hasznosabb a weboldal URL-jéből kinyerni az információt, mert a szerver IP címe mindig változik attól függően, hogy épp hol, milyen router alatt van tesztelve a szoftver. Az API hívás az /*api/data?unique\_ids* endpoint-ról kéri az adatokat.

**Render**

A render-ben rejlik az oldalválasztás logikája.

render() {

const { sensor\_id\_array, SERVER\_API\_ADDR } = this.state;

return (

<React.Fragment>

<NavBar ids={sensor\_id\_array} />

<Routes>

<Route

path="/sensor/:id"

element={<Charts SERVER\_API\_ADDR={SERVER\_API\_ADDR} />}

/>

<Route path="/" element={<TextRotator username={"User"} />} />

</Routes>

</React.Fragment>

);

}

A navigációs sávot minden esetben be kell tölteni, de a többi komponenst már nem. A chart-ok csak a */sensor/x* elérési útvonal esetén töltenek be, ahol *x* egy szám. A *TextRotator* a weblap gyökér elérési útja esetén tölt be.

**App.css**

Az App-hoz tartozik egy CSS file, ami nagyon egyszerű.

.table {

table-layout: fixed;

}

.navbar {

margin-bottom: 0px;

}

A *table-layout* beállítása fix táblázat elrendezést határoz meg. A *navbar*-ok esetében alulról minden margó el van tüntetve.

### Inicializás

A React project létrehozásakor az npm generált egy alap HTML file-t és az alábbi kód egy részletét (BrowserRouter és QueryClientProvider nélkül). A generált HTML file-ban van egy külön *root* azonosítójú elem, ennek az inicializálását oldotta meg automatikusan az npm a generáláskor. A megfelelő használathoz ezen a kódon némileg változtatni kellett.

const queryClient = new QueryClient();

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(

<BrowserRouter>

<QueryClientProvider client={queryClient}>

<App />

</QueryClientProvider>

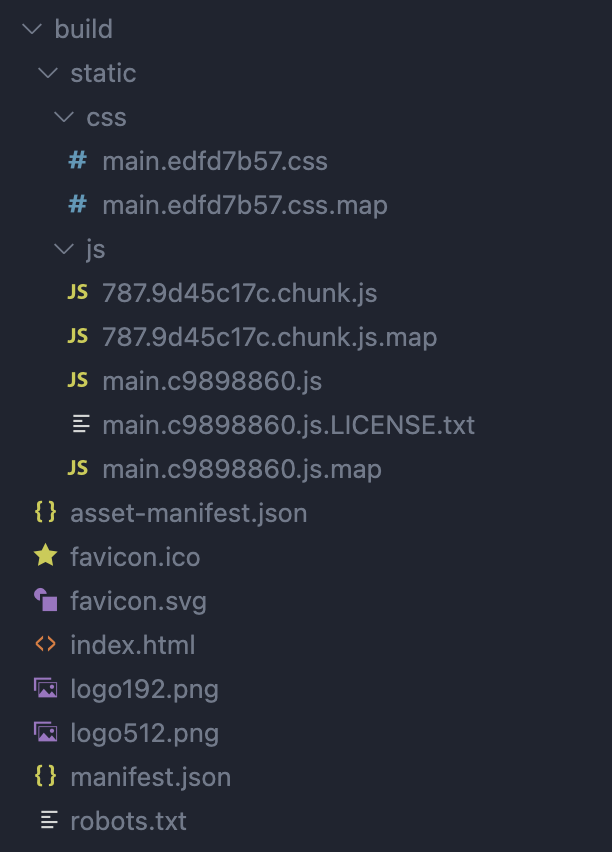
</BrowserRouter>

);

Az App gyökér komponens a *BrowserRouter* és *QueryClientProvider* komponensekkel van körülvéve. Ez a React Router (átirányítás, linkek stb.) és a QueryClient (cacheléses API hívás) használatát biztosítja az App számára.

### Build file-ok

A React project build-elését az *npm run build* paranccsal lehet megtenni. Az így generált filestruktúrát a 20. ábra tudja a legjobban bemutatni.



20. ábra

Látszik, hogy valóban 1 oldalas a React weboldal, hisz csak 1 HTML file jön létre. Minden file a *build* könyvtár alá generálódik. A CSS file-ok a *static/css* könyvtárba, a JavaScript file-ok a *static/js* könyvtárba kerülnek. Minden egyéb file közvetlen a *build* könyvtár alá kerül.

# Komponensek közvetlen kapcsolatai

Az előző fejezetekben bemutattam a projectet alkotó egységek működését. Ugyan, ezen leírások tartalmaznak minden információt, amelyek szükségesek az interface-k megértéséhez, nem tértem ki külön arra, hogy hogyan, és miért is tudnak ezek egymással kommunikálni. Ebben a fejezetben szeretnék rávilágítani a komponensek közötti közvetlen kapcsolatokra, azaz csak azokra, amely két egymással szomszédos komponens között van és nem halad át másik –a szakdolgozatban tárgyalt– egységen.

## Adatgyűjtő és Szerver

Az Adatgyűjtő ugyebár a Szerver részére minden adatot elküld. Egy adatcsomag egyszerre több számértékből áll, így valahonnan tudnia a kell a szervernek, hogy melyek az összetartozó adatok. Erre azt a protokollt találtam ki, hogy az Adatgyűjtőnek minden logikailag egybetartozó adatsorozat után egy *„q”* lezáró karaktert kell küldenie. Így már a TCP szerver tudja értelmezni az adatokat. Az Adatgyűjtőnél *a SendDataWithESP* task, a TCP szervernél az *Adatkezelő segédkönyvtár* és a *thread\_socket\_handle* szál felel ezen protokoll megvalósításáért.

## Szerver és Weblap

A Szerver és a Weblap kapcsolata kétrétű. A Szerver a Weblap számára adatokat és file-okat biztosít. A kérdés, hogy ezeket hogyan kell kérni, és hogyan kell értelmezni a visszakapott információkat. Minden esetben a Szerver IP címére szükséges küldeni a kéréseket.

### API

Az API-n keresztül tud a Weblap 2 féle képpen adatot kérni. Mindkét esetben az endpoint-ok gyökere az */api/data* URL-en érhető el.

#### Id endpoint

Ez az endpoint az */api/data?id=x* URL-en érhető el, GET kéréssel. Itt a Szerver az Adatbázisból azon hőmérsékleti és páratartalmi adatokat kéri le, amelyek az *x* *sensor\_id*-jú Adatgyűjtőhöz tartoznak. Ezeknek az adatoknak a minimumát is lekérdezi, majd a következő formátum szerint tér vissza:

{ "temp" "humid" "min\_humidity" "min\_temperature" }

Az is hozzátartozik, hogy a *temp* és a *humid* címkékhez tartozó adattömbök tovább bonthatóak a következők szerint:

temp =[{date: x, value: y}, ...]

humid =[{date: x, value: y}, ...]

, így JavaScriptben az ettől az endpoint-tól kapott adatok kinyerése

Const { temp, humid, min\_humidity, min\_temperature } = endpoint\_data

szerint történhet a legegyszerűbben.

#### Unique\_ids endpoint

Ennek az endpoint-nak az URL-je */api/data?unique\_ids*, amely ugyan úgy GET kéréssel érhető el. A Szerver itt az Adatbázisban megtalálható minden egyedi *sensor\_id* értéket visszaküld egy tömbben a *sensor\_id\_array* címkével ellátva, úgyhogy az optimális fogadása az adatnak a frontend oldalon:

Const { sensor\_id\_array } = endpoint\_data

Ezzel tulajdonkéépen az az információ kérhető el, hogy milyen *sensor\_id*-jú Adatgyűjtők vannak.

### Webserver

A Szerver Webszerver része a nevéből is adódóan, webszerverként funkcionál. A frontend a Szerver IP címén tud webfile-okat kérni. A Szerver úgy tudja ezt kiszolgálni, hogy ismeri a frontend által felhasznált filerendszer struktúráját. Ehhez ugyan tudnia kell a Webszervernek, hogy hol vannak a generált build file-ok, de ha ez megvan, akkor hiánytalanul minden HTTP file kérést ki tud szolgálni.

## Szerver és Adatbázis

Ez a kapcsolat annyiból áll, hogy az Adatbázisban regisztrálva van egy felhasználó a Szerver számára. A megfelelő felhasználónévvel és jelszóval képes a Szerver csatlakozni.

# Komponensek közvetett kapcsolatai

Ezen fejezet célja azon összefüggéseket feltárni, ami két olyan komponens között áll fent, amelyek nem állnak közvetlen kapcsolatban egymással.

## Adatgyűjtő és Weblap

Az utolsó nem tárgyalt kapcsolat az Adatgyűjtő és a Weblap között van. Itt, a kérdés: hogyan tudja a Weblap az adott szenzor adatait megjeleníteni? A válasz abban rejlik, hogy minden szenzor el van látva egy egyedi azonosítóval, ami az adatbázisban a *sensor\_id*. A Weblap az */api/data?id=x* endpoint-on tudja elkérni egy adott *sensor\_id*-jű Adatgyűjtő adatait, így megválaszolván a kérdést.

# Értékelés

A szakdolgozat végére érvén két szempontból szeretném értékelni a szakdolgozatom. Egyrészről, hogy szakmailag mennyire helytálló és igényes a megoldás, másrészről, hogy milyen módokon lehetne az alkotáson javítani, illetve továbbfejleszteni.

## Elemzés

Véleményem szerint az általam megvalósított mérnöki mű megfelel az elvárható minimális kritériumoknak és egy korabeli problémára ad megoldást, korszerű eszközökkel. Jól tagolt és kellően logikusan részekre bontott a továbbfejleszthetőséget és a modularitást figyelemmel tartva. Az egységek interface-jei egyszerűen és könnyen kezelhetőek mind mérnöki, mind felhasználói szemszögből. Ezek mind elvárhatóak és hozzátartoznak egy jól átgondolt műszaki alkotáshoz.

A sok pozitívuma ellenére sajnos hagy kivetnivalót. A legnagyobb hiányossága a biztonság negligálása. A fejlesztés során egy ideális felhasználóközönséget képzeltem el, akik tökéletesen tisztában vannak azzal, hogy hogyan kell használni. Ez természetesen nem reális. Nem megfelelő használat esetén előfordulhatnak kisebb hibák, illetve nincs védve kiberbiztonsági szempontból. Szerencsére semmilyen maradandó kár nem okozható a rendszerben, pusztán nem megfelelő használat esetén romlik a felhasználói élmény.

## Továbbfejleszthetőség

A fejlesztés során külön figyelemmel voltam a továbbfejleszthetőség megtartására. Számos további funkciókkal lehet kiegészíteni az alkotást.

Talán a legegyértelműbb a weboldal fejlesztése. A chart-okkal lehetne több mindent csinálni, nem csak nagyítani és mozgatni. Lehetne több Adatgyűjtő adatait is összehasonlítani, egymásra illetve egymás mellé rakni, személyre szabni a chart-okat színváltoztatással és egyéb paraméterekkel stb.

Lehetne felhasználói fiókokat létrehozni és kezelni. Ehhez már szükséges lenne a Szerver és az Adatbázis módosítása, de csak egy új táblára és egy új endpoint-ra lenne szükség.

Egy funkció, amit implementálni szerettem volna, de nem jutott rá idő, az az élő adatkövetés. Egy igazán ergonomikus funkció lett volna, hogy élőben lehet nézni a weboldalon, hogy melyik Adatgyűjtő mit mér.

A fent említetteken felül számtalan további funkcióval lehetne még bővíteni a rendszert például nem csak hőmérséklet és páratartalom mérése, GPS modul illesztése a helymeghatározáshoz, ergonomikusabb hibakezelő szoftver a Touchgfx komponenshez. A további lehetőség kitalálását az olvasó fantáziájára bízom.

# Irodalomjegyzék

1. C programozási nyelv angol nyelvű Wikipedia leírása (2023.10.09): <https://en.wikipedia.org/wiki/C_(programming_language)>
2. C++ programozási nyelv angol nyelvű Wikipedia leírása (2023.10.09): <https://en.wikipedia.org/wiki/C%2B%2B>
3. Python programozási nyelv hivatalos honlapja (2023.10.09): [https://www.python.org](https://www.python.org/)
4. HTML programozási nyelv angol nyelvű Wikipedia leírása (2023.10.09): <https://en.wikipedia.org/wiki/HTML>
5. CSS programozási nyelv angol nyelvű Wikipedia leírása (2023.10.09): <https://en.wikipedia.org/wiki/CSS>
6. JavaScript programozási nyelv angol nyelvű Wikipedia leírása (2023.10.09): <https://en.wikipedia.org/wiki/JavaScript>
7. Document Object Model angol nyelvű Wikipedia leírása (2023.10.09): <https://en.wikipedia.org/wiki/Document_Object_Model>
8. React könyvtár hivatalos honlapja (2023.10.10): <https://react.dev>
9. Felhasználói felületek (UI) angol nyelvű Wikipedia leírása (2023.10.09): <https://en.wikipedia.org/wiki/User_interface>
10. Bootstrap hivatalos honlapja (2023.10.10): <https://getbootstrap.com>
11. FontAwesome hivatalos honlapja (2023.10.10): [https://fontawesome.com](https://fontawesome.com/)
12. FreeRTOS hivatalos honlapja (2023.10.10): [https://www.freertos.org](https://www.freertos.org/)
13. CMSIS\_RTOS2 dokumentációja (2023.09.18): <https://arm-software.github.io/CMSIS_5/latest/RTOS2/html/index.html>
14. MySQL hivatalos honlapja (2023.10.10): [https://www.mysql.com](https://www.mysql.com/)
15. Git hivatalos honlapja (2023.10.10): [https://git-scm.com](https://git-scm.com/)
16. GitHub hivatalos honlapja (2023.10.10): [https://github.com](https://github.com/)
17. VS Code hivatalos honlapja (2023.10.10): [https://code.visualstudio.com](https://code.visualstudio.com/)
18. Node.js hivatalos honlapja (2023.11.06): <https://nodejs.org/en>
19. STM32CubeIde hivatalos weboldala (2023.09.05): <https://www.st.com/en/development-tools/stm32cubeide.html>
20. STMicroelectronics hivatalos honlapja (2023.09.23): <https://www.st.com>
21. TouchGFX hivatalos weboldala (2023.09.05): <https://www.st.com/en/development-tools/touchgfxdesigner.html>
22. MySQL Workbench hivatalos weboldala (2023.10.16): <https://www.mysql.com/products/workbench/>
23. Roger Meier’s Freeware, CoolTerm honlapja (2023.09.05): <http://freeware.the-meiers.org>
24. PuTTY hivatalos honlapja (2023.10.09): <https://www.putty.org>
25. Espressif esptool.py dokumentációja (2023.09.10): <https://docs.espressif.com/projects/esptool/en/latest/esp32/>
26. STM32H750B-DK mikrokontroller hivatalos weboldala (2023.09.18): <https://www.st.com/en/evaluation-tools/stm32h750b-dk.html#overview>
27. STM32H750B-DK STMOD+ fanout board ismertető letöltő linkje (2023.09.18): <https://www.st.com/resource/en/user_manual/um2695-stmod-fanout-expansion-board-for-stm32-discovery-kits-and-evaluation-boards-stmicroelectronics.pdf>
28. ESP8266-01 WIKI jellegű ismertető(2023.10.18): [https://nurdspace.nl/ESP8266#Uploading\_code](https://nurdspace.nl/ESP8266" \l "Uploading_code)
29. SHT31 SEN0385 Szenzor adatlap (2023.10.18): <https://dfimg.dfrobot.com/nobody/wiki/88b31350da4f54d00989c74c6fa392f7.pdf>
30. Dfrobot WIKI leírás az SHT31 SEN0385 Szenzorról (2023.10.18): <https://wiki.dfrobot.com/SHT31_Temperature_Humidity_Sensor_Weatherproof_SKU_SEN0385>
31. „Query” fogalom magyarázat (2023.11.28): <https://www.techtarget.com/searchdatamanagement/definition/query#:~:text=A%20database%20query%20is%20either,other%20forms%20of%20data%20manipulation>.
32. Python mysql-connector könyvtár dokumentációja (2023.11.28): <https://pypi.org/project/mysql-connector-python/>
33. Python socket könyvtár dokumentációja (2023.11.28): <https://docs.python.org/3.11/library/socket.html>
34. Python threading könyvtár dokumentációja (2023.11.28): <https://docs.python.org/3/library/threading.html>
35. Python flask könyvtár dokumentációja (2023.11.28): <https://flask.palletsprojects.com/en/3.0.x/>
36. Python gunicorn HTTP szerver hivatalos honlapja (2023.11.28): <https://gunicorn.org>
37. Espressif ESP AT-Command dokumentáció (2023.09.06): <https://docs.espressif.com/projects/esp-at/en/release-v2.2.0.0_esp8266/index.html>
38. ESP8266 felhasznált firmware (2023.09.05): <https://github.com/CytronTechnologies/esp-at-binaries/blob/main/Cytron_ESP-01S_AT_Firmware_V2.2.0.bin>
39. TCP FIN flag ismertető (2023.12.02): <https://www.geeksforgeeks.org/tcp-flags/>
40. STM32H750B-DK user manual (2023.09.11): <https://www.st.com/resource/en/user_manual/um2488-discovery-kits-with-stm32h745xi-and-stm32h750xb-mcus-stmicroelectronics.pdf>
41. Controllerstech Touchgfx billentyűzet mintha, fileokkal (2023.09.30): <https://controllerstech.com/touchgfx-7-how-to-implement-on-screen-keyboard/>
42. DevExtreme Zooming and Panning API (2023.11.05): <https://js.devexpress.com/React/Demos/WidgetsGallery/Demo/Charts/ZoomingAndScrollingAPI/MaterialBlueLight/>
43. React Router 6.20.1. dokumentáció (2023.12.05): <https://reactrouter.com/en/6.20.1>
44. Axios 1.6.2. dokumentáció (2023.12.05): <https://www.npmjs.com/package/axios/v/1.6.2>
45. Bootstrap táblázat példa (2023.11.11): <https://getbootstrap.com/docs/4.0/content/tables/#examples>
46. Bootstrap Navbar példa (2023.11.13): <https://getbootstrap.com/docs/4.0/components/navbar/#nav>
47. FontAwesome nagyító ikon (2023.11.13): <https://fontawesome.com/icons/magnifying-glass-chart?f=classic&s=solid&rt=flip-horizontal>