Ipar4TK IoT Platform

Kerezsi Zoltán – M0W4LW – 2023.06.07

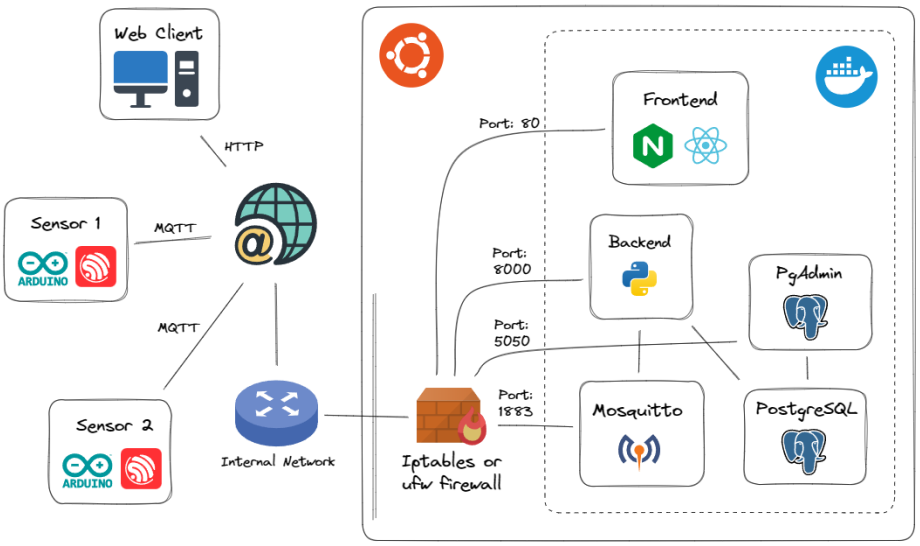
Bevezetés

A projekt célja, hogy készítsünk egy szoftvert amelyik képes valós időben fogadni, feldolgozni, eltárolni és megjeleníteni adatokat amelyek különféle szenzorokból érkeznek.

A fejlesztés során nagyban támaszkodtam a Docker Compose nevű eszközre, amelynek segítségével konténerizált folyamatokat lehet futtatni egy YAML fájl alapján, amely a teljes környezetet leírja, így később is reprodukálható a teljes szoftvercsomag.

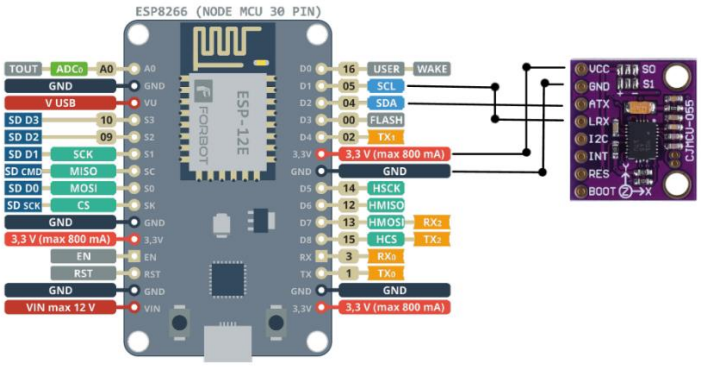
Architekúra

A Docker környezetben futó folyamatok a szaggatott vonalon belül helyezkednek el. Ezeknek a portjait kell a gazda operációs rendszer, majd a belső hálózaton kiengedni az internet felé, hogy a szenzorok küldhessenek az MQTT brókernek adatokat, hogy be tudjuk tölteni a kezelőfelületet, illetve hogy láthassuk a PgAdmin felületét amin keresztül teljeskörűen hozzáférhetünk a tárolt adatokhoz.



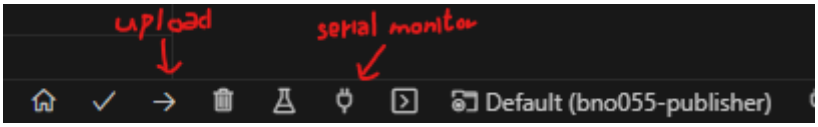
Mikrovezérlő

Egy BNO055 szenzort használtam az adatok forrásául. Az ebből kilvasott adatokat a mikrovezérlő egy FFT algoritmussal feldolgozza, majd a forráskódban látható módon elküldi az MQTT brókernek, így az üzenet hasznos tartalma mindig 1024 bájt hosszú. A szenzort a mikrovezérlővel az alábbi illusztráción látható módon kötöttem össze.

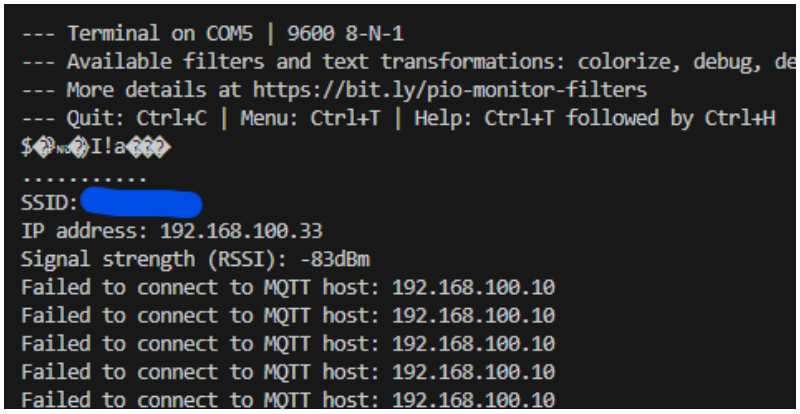


Kitelepítés

A projekt mappáját letöltve nyissuk meg a bno055-publisher mappát VSCode-ban, ahol előzetesen feltelepítettük a PlatformIO bővítményt. A forrásfájloknál hozzunk létre egy Secrets.h header fájl-t amibe át kell másolni a Secrets.example.h fájl tartalmát és értelemszerűen módosítani a kívánt értékekkel. Miután megvártuk, hogy a fejlesztőkörnyezet automatikusan telepítse a megfelelő csomagokat elmúlik minden hibajelzés.



Ezután feltölthetjük a kódot az ESP8266-ra. Ha minden jól ment, a soros monitorra kapcsolva láthatjuk amint a mikrovezérlő csatlakozni próbál a hálózathoz, majd az MQTT brókerhez.



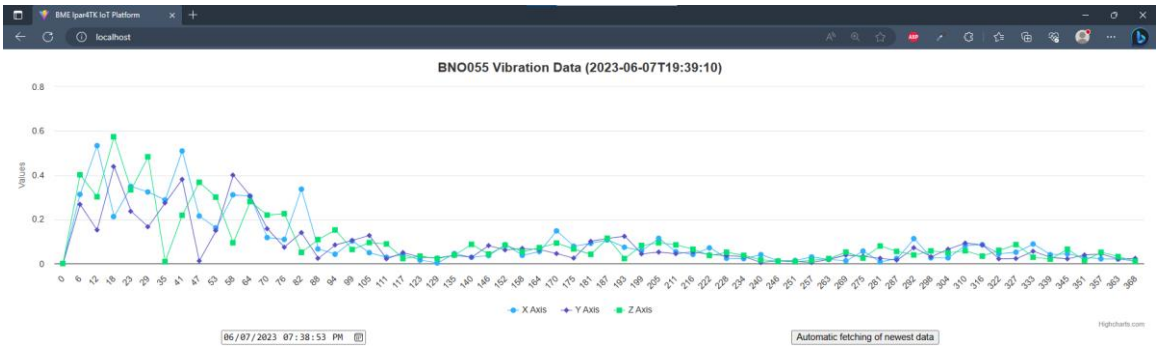
Ha ez megvan, nyissuk meg a projekt gyökérmappáját egy terminálban, ahol a <docker compose up> parancsot futtatva minden szükséges Docker képfájl letöltődik vagy megépül, ahogyan azt a docker-compose.yml fájlban definiáltuk.

Első indításnál a PostgreSQL adatbázis indulása hosszabb időbe telhet. Ekkor a backend folyamat hibát dobva leáll, mivel nem tud csatlakozni a még el nem indult adatbázishoz. Egy idő után az adtabázis is elkezdi válaszolni a backend-nek, így beáll az egyensúly. A rendszer önmagát “gyógyítja”, ha egy folyamat nem tervezett módon leáll, a Docker Compose gondoskodik az újraindításról.

```
admin@local MINGW64 ~/workspace/bmetk-final (master)
$ docker compose up
[+] Running 5/0
 - Container bmetk-final-postgres-1   Created
 - Container bmetk-final-mosquitto-1  Created
 - Container bmetk-final-backend-1    Created
 - Container bmetk-final-frontend-1   Created
 - Container bmetk-final-pgadmin-1    Created
Attaching to bmetk-final-backend-1, bmetk-final-frontend-1, bmetk-final-mosquitto-1, bmetk-final-pgadmin-1, bmetk-final-postgres-1
bmetk-final-mosquitto-1  | 2023-06-07T19:30:00: mosquitto version 2.0.15 starting
bmetk-final-postgres-1   |
bmetk-final-postgres-1   | PostgreSQL Database directory appears to contain a database; Skipping initialization
bmetk-final-postgres-1   |
bmetk-final-postgres-1   | 2023-06-07 19:30:01.325 UTC [1] LOG:  starting PostgreSQL 15.2 on x86_64-pc-linux-musl, compiled by gcc (Alpine 12.2.1_git20220924-r4) 12.2.1 20220924, 64-bit
bmetk-final-postgres-1   | 2023-06-07 19:30:01.325 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
bmetk-final-postgres-1   | 2023-06-07 19:30:01.325 UTC [1] LOG:  listening on IPv6 address "::", port 5432
bmetk-final-postgres-1   | 2023-06-07 19:30:01.332 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
bmetk-final-postgres-1   | 2023-06-07 19:30:01.372 UTC [24] LOG:  database system was shut down at 2023-06-07 19:29:51 UTC
bmetk-final-postgres-1   | 2023-06-07 19:30:01.426 UTC [1] LOG:  database system is ready to accept connections
bmetk-final-pgadmin-1    | postfix/postlog: starting the Postfix mail system
bmetk-final-backend-1    | 2023-06-07 19:30:01,533 [INFO]: Starting backend service on http://localhost:8000
bmetk-final-backend-1    | 2023-06-07 19:30:01,534 [DEBUG]: Using selector: EpollSelector
bmetk-final-backend-1    | 2023-06-07 19:30:01,698 [INFO]: Successfully connected to Mosquitto
bmetk-final-frontend-1   | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
bmetk-final-frontend-1   | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
bmetk-final-frontend-1   | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
bmetk-final-frontend-1   | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
bmetk-final-frontend-1   | 10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf differs from the packaged version
bmetk-final-frontend-1   | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
bmetk-final-frontend-1   | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
bmetk-final-frontend-1   | /docker-entrypoint.sh: Configuration complete; ready for start up
bmetk-final-frontend-1   | 2023/06/07 19:30:01 [notice] 1#1: using the "epoll" event method
bmetk-final-frontend-1   | 2023/06/07 19:30:01 [notice] 1#1: nginx/1.24.0
bmetk-final-frontend-1   | 2023/06/07 19:30:01 [notice] 1#1: built by gcc 12.2.1 20220924 (Alpine 12.2.1_git20220924-r4)
bmetk-final-frontend-1   | 2023/06/07 19:30:01 [notice] 1#1: OS: Linux 5.10.102.1-microsoft-standard-WSL2
bmetk-final-frontend-1   | 2023/06/07 19:30:01 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
bmetk-final-frontend-1   | 2023/06/07 19:30:01 [notice] 1#1: start worker processes
```

Frontend

A localhost:80 címet megnyitva láthatjuk a frontend-et. A weboldal másodpercenként lekérdezi a backend-től a legfrissebb szenzor adatot, majd megjeleníti egy Highcharts diagramon. A jobb oldali gombra kattintva lehetőségünk adódik egy múltbéli adat megjelítésére is, amit a HTML input mezővel tudunk kiválasztani. A diagram X tengelyén a frekvenciák, az Y tengelyén pedig a magnitúdójuk látható.



Adattárolás

Kezdetben az InfluxDb nevű terméket akartuk használni a hozzá való Telegraf megjelenítő és adatgyűjtő eszközzel, viszont a hiányos dokumentáció és a váratlan adattárolási nehézségek miatt végül a PostgreSQL adatbázist használtam. A források szekciónál belinkelt cikk alapján egy BRIN indexet helyeztem a rekordok időbélyegére. A PostgreSQL remekül támogatja a félstrukturált adattárolást, ezért egy táblának nem kötelező egyedi kulcsot megadni. A szenzorból érkező adatot bináris JSON formátumban tárolja el, így a formátuma nincs megkötve.

```
create_table = '''
CREATE TABLE IF NOT EXISTS sensor_data(
    _timestamp TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP,
    _sensor VARCHAR NOT NULL,
    _measurement VARCHAR NOT NULL,
    _value JSONB NOT NULL
);
'''
await cur.execute(create_table)
create_index = '''
CREATE INDEX IF NOT EXISTS _timestamp_index ON sensor_data USING BRIN(_timestamp);
'''
await cur.execute(create_index)
```

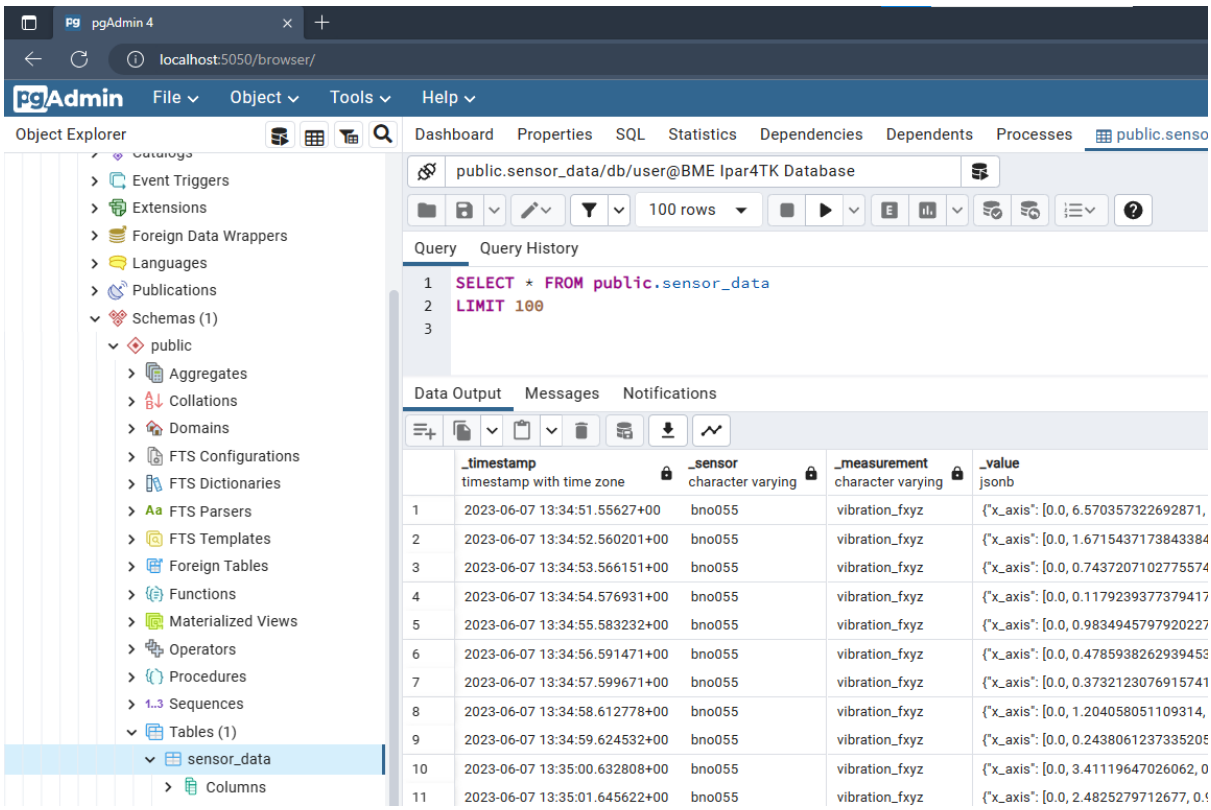
A lekérdezés a következő módon történik, mindig az időben legközelebbi adatot fogja válaszul adni:

```
command = '''
SELECT s._timestamp, s._sensor, s._measurement, s._value
FROM sensor_data s
WHERE s._timestamp <= %s
    AND s._sensor = 'bno055'
    AND s._measurement = 'vibration_fxyz'
ORDER BY s._timestamp DESC
LIMIT 1;
'''
await cur.execute(command, (timestamp,))
```

Idő paraméter nélküli HTTP kérésre a backend mindig a legfrissebb adattal válaszol:


```
{"_timestamp": 1686167500000, "_sensor": "bno055", "_measurement":
"vibration_fxyz", "_value": {"x_axis": [0.0, 0.757823646068573,
0.18709853291511536, 0.5329012274742126, 0.3427949845790863, 0.34256
0.25639617443084717, 0.3498918414115906, 0.1701989471912384, 0.45921
0.2199486792087555, 0.19839216768741608, 0.29521963000297546, 0.3096
0.15008357167243958, 0.23796355724334717, 0.12934257090091705, 0.131
0.07337858527898788, 0.08889263868331909, 0.0949045866727829, 0.0443
```


A PgAdmin-felületére bejelentkezve hozzáférhetünk az összes adathoz és bármilyen query-t futtathatunk, vagy kimenthetjük a tábla tartalmát csv formátumba. A bejelenkezéshez szükséges email és jelszó szintén a docker-compose.yml fájlban található.



Továbbfejlesztés és bővítés

A backend-hez tetszés szerint hozzáadható új szenzor feldolgozása is. Ekkor az aiomqtt_coro függvényben fel kell iratkozni az új topic-ra és a match-case-ben hozzáadni az új függvényt ami az adott szenzor adatait eltárolja majd az adatbázisban. Ezután hozzáadhatunk új HTTP végpontontot és adatbázis-lekérdezést is, azaz az egész rendszeren végig kell vezetni az adatot hogy végül megkapja frontend.

```
async with asyncio_mqtt.Client(hostname=mqtt_hostname,
                               port=mqtt_port,
                               password=mqtt_password,
                               username=mqtt_username) as client:
    await client.subscribe("bno055")
    app["mqtt_healthy"] = True
    logger.info(
        f'Successfully connected to Mosquitto')
    async with client.messages() as messages:
        async for message in messages:
            buffer = typing.cast(bytes, message.payload)
            match f'{message.topic}':
                case 'bno055':
                    await insert_bno055_data(pool, buffer)
```

A Dockerfile-ok FROM verzióit lehet később növelni, illetve a docker-compose.yml fájlban is ugyanezt meg lehet tenni. Ugyanez vonatkozik a függőségek verzióira (requirements.txt, package.json), ami viszont minimális kódváltoztatással járhat.

Források

https://docs.aiohttp.org/en/stable/web_advanced.html#cleanup-context

https://www.npmjs.com/package/highcharts-react-official

https://github.com/Tinyu-Zhao/FFT

https://pypi.org/project/asyncio-mqtt/

https://aiopg.readthedocs.io/en/stable/index.html

https://docs.docker.com/get-started/08_using_compose/

https://www.crunchydata.com/blog/postgresql-brin-indexes-big-data-performance-with-minimal-storage