ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Alba Regia Műszaki Kar
Természettudományi és Szoftvertechnológiai Intézet

SZAKDOLGOZAT

Hangmérő rendszer tervezése és megvalósítása online megjelenítéssel

OE-AMK

2024

Hallgató neve: Bedő Alex István

Hallgató törzskönyvi száma: T000625./FI12904/A-M

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbudai Egyetem
Alba Regia Műszaki Kar
Természettudományi és Szoftvertechnológiai Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: Bedő Alex István
Szakdolgozat száma: SZD20042009013216
Törzskönyvi száma: T000625./FI12904/A-M **Neptun kódja:** [redacted]

Szak: Mérnökinformatikus
Specializáció: Big Data és üzleti intelligencia specializáció

A dolgozat címe: Hangmérő rendszer tervezése és megvalósítása online megjelenítéssel
A dolgozat címe angolul: Design and Implementation of a Sound Measurement System with Online Interface
A feladat részletezése: Környezetünkben jelentősen megnövekedett a hangszennyezettség. Ez igaz akár az ipari környezetre, de a hétköznapi életünkben is egyre nagyobb mértékben van jelen. Így ennek mérése, nyomon követése fontos feladat. A fellelhető eszközök drágák, így ezért egy pillanatnyi állapotfelmérést valósítanak meg velük.
A feladatmegoldás során a szakdolgozó valósítson meg egy olyan rendszert, amely lehetővé teszi hangerő szintek folyamatos mérését egy mikrofon segítségével, azok előkészítését feldolgozásra. Az egyes mérőpontok és a központi kiszolgáló között online kommunikációt tegyen lehetővé, és így a feldolgozott adatokat tegye elérhetővé webes felületen keresztül. A feladatmegoldás során készítse el mind a mérőeszközt, mind a hozzá kapcsolódó szoftvereket.
A mérőeszköz szoftveres megoldásakor használjon Python nyelvet. A szerveroldali megjelenítéskor a CSS, JavaScript, PHP nyelveket használja.

Intézményi konzulens neve: Gugolya László

A kiadott téma elévülési határideje:2024. 07. 10.
Beadási határidő: 2024. 05. 15.

A szakdolgozat: Nem titkos.

Kiadva: Székesfehérvár, 2024. 05. 14. P.H. ..... Intézetigazgató

A dolgozatot beadásra alkalmasnak találom: 2022. 05. 15.

..... belső konzulens

..... külső konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Alba Regia Műszaki Kar
Természettudományi és Szoftvertechnológiai Intézet

HALLGATÓI NYILATKOZAT

Alulírott Bedő Alex István kijelentem, hogy a dolgozat saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült dolgozatban található eredményeket az Óbudai Egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja, a titkosításra vonatkozó esetleges megkötések mellett.

Kelt: Székesfehérvár, 2024.05.14.

hallgató aláírása

Tartalomjegyzék

1. Bevezetés

Napjainkban egyre inkább szem elé kerül a zajszennyezettség kérdése, és az ezzel kapcsolatos aggodalom megjelenik több helyen. Munkavédelmi szempontból fontos kérdés minden ipari gyárban és műhelyben. Megtalálható a városi élet során több helyen, de a Panellakásban élők esetében különösen releváns lehet. Ahol a természetes állatvilág élőhelyének közelében történnek különböző munkafolyamatok, ott fontos tényező lehet a zajszennyezettség, környezetvédelmi szempontból.

Ennek a figyelésére rengeteg mérőeszköz található a piacon, a különböző zajmérő eszközök látványa nem okoz senkinek meglepetést. A zajszint szabályozására számos követelményt definiáltak, és ellenőrzik is azokat.

A rendelkezésre álló megoldások mennyiségének ellenére mégis azt gondolom van olyan résztvevője ennek a társadalmi helyzetnek, akinek a helyzetén lehet javítani egy (leginkább) szoftveres megoldással. Azok számára, akik felelősséget vállalnak az említett követelmények betartásáért, nem állnak rendelkezésre olyan megoldások, amik lehetővé tennék a folyamatos megfigyelést. A zajszint folyamatos mérése lehetővé teheti a cégeknek, hogy időben fel tudjanak készülni és tisztában legyen a kritikus helyszínekkel és időszakaszokkal, ha a zajszint elér egy olyan szintet, ami valamilyen extra kötelességgel jár, például ha hallásvédő eszközöket kell biztosítsanak a munkavállalóiknak azt időben megtehetik. Ha olyan vád éri őket, hogy átlépték valamilyen zajküszöböt, akkor ennek a kivizsgálásában egy jó kiindulási pont lehet, ha van egy rendszer, ami meg tudja mondani, hogy a vizsgált időpontban volt-e kiugró érték a zajszintben, és ha igen akkor megközelítőleg hol volt ennek a forrása.

Egy ilyen rendszer könnyebbséget jelenthet az olyan cégeknek is, akik ilyen mérésekkel foglalkoznak, ha ezzel le lehet csökkenteni az órák számát, amikor valakinek jelen kell lenni a mérés helyszínén.

Szakdolgozatom célja egy olyan szoftver rendszer leprogramozása, ami lehetővé teszi zajmérő eszközök telepítését, és az ezekből az eszközökből kinyert adat egy webes felületen való megjelenítését. A szoftvernek az üzleti logika és adatfeldolgozó részét leginkább C# nyelven írt .NET kód alkotja, az adatok eltárolására egy MySQL adatbázist használtam, a felhasználói felület megírásához pedig Vue.js-t használtam.

2. Probléma megfogalmazás, feladatspecifikáció

A cél egy olyan rendszer megalkotása, amely lehetővé teszi, hogy a felhasználó több zajszint mérő eszközt helyezzen el, amik egy-egy mikrofon segítségével folyamatosan képesek előállítani ezt az adatot. Az ezek által generált adatot, egy központi webes felületen vissza tudja nézni grafikusan, továbbá élő adatokat is elérhetővé tesz.

Funkcionális követelmények

- Zajszintmérés: A rendszernek képesnek kell lennie zajszintet mérni, olyan kis eszközökkel, amiből akár többet is el lehet helyezni egy területen.
- Élő adatok megjelenítése: A rendszer meg kell tudja jeleníteni a pillanatnyi adatokat a webes felületen.
- Történelmi adat: A méréseket a rendszer el kell tudja tárolni hosszú távon.
- Vizuális megjelenítés: Az adatokat meg kell tudja jeleníteni a rendszer egy vizuális formában, mint például egy grafikonon vagy egy térképen.
- Több eszköz : A rendszernek támogatnia kell több eszköz együttes működését, és ezeknek az adatait fel kell tudja dolgozni.
- Frekvenciafelbontás: A Zajszintet fel kell tudja bontani a rendszer, legalább mély, magas és közép hangokra.

Nem funkcionális követelmények

- Reszponzív: A rendszer mobil eszközöknek is megfelelő megjelenést kell nyújtson.
- Felhasználóbarát: A felületnek felhasználóbarátnak kell lennie.
- Megbízhatóság: A rendszer megbízhatóan és stabilan kell működjön több eszköz esetén is.

3. Megvalosíthatósági tanulmány

Szakdolgozatom készítése előtt megvizsgáltam az alternatívákat. A legelterjedtebb a kézben tartható zajmérő készülékek használata. Ebből rengeteg félét találhatunk,

Jellemzően van rajtuk egy LCD kijelző, amin láthatjuk a pillanatnyi mért zajszintet. A profibb eszközök között előfordul olyan, ami egy memóriakártyára mentik a mérési adatokat és később kielemezhetőek egy speciális szoftver használatával.

Emellett fellelhetőek olyan projektek, amik közelebb állnak az én munkámhoz, hasonló hardverre épülnek és zajszintet mérnek, azonban hasonlóan az előbbi eszközökhöz, egy kijelzőre jelenítik meg az adatokat, vagy a saját háttértárjukra mentik ezeket az adatokat.

Minden alternatíva, amit megvizsgáltam szükségessé tette, hogy valaki aktívan jelen legyen a méréshez, ez bizonyos esetekbe elegendő lehet, azonban egy folyamatos méréshez és/vagy egy több pontban történő méréshez kevés.

Ezeket figyelembe véve, úgy gondolom, hogy egy olyan rendszer ami könnyen telepíthető zajmérő eszközöket köt össze egy központi adatbázissal és egy webes megjelenítési felülettel jelentősen megkönnyíti az ilyen adatok hozzáférhetőségét.

4. Tervezés

4.1 Szerkezet

A probléma megoldására egy három szoftverkomponensből álló rendszert terveztem meg. Az első a kliensszoftver a zajmérések elvégzésére és küldésére. A második az adatok fogadására és az adatbázisba való feltöltésére, valamint kérésre az adatbázisból való lekérésre szolgáló API. A harmadik a webes felület, ami lehetővé teszi a felhasználóknak, hogy megtekintsék az adatokat egy értelmezhető vizuális formátumban. Úgy gondolom, hogy a rendszer csak úgy tud megfelelő stabilitással üzemelni, ha "on-premises" rendszerként működik. Felhős megoldásként nem lenne fenntartható egy lehetséges felhasználó számára, tekintettel a forgalom mennyiségére, amit ez okozna az internet vonalán feleslegesen.

4.2 Adatbázis terv

A rendszer működéséhez szükséges eltárolni a Mérési adatokat valamilyen formában. Ehhez 3 féle adatbázis-kezelő rendszert vizsgáltam meg.

4.2.1 NoSQL

Az eredeti elképzelés szerint MongoDB adatbázis-kezelő rendszert használt volna a rendszer, mivel a MongoDB alapvetően JSON dokumentumokat fogad el és ad vissza, és mivel a REST API is JSON dokumentumokkal kommunikál, így jelentősen le lehetett volna csökkenteni az API szerepét, megkönnyítve a tervezési folyamatát. Az első nagyobb prototípus készítése közben egy technikai problémába ütköztem, és a szerveren kompatibilitási okok miatt nem futott az adatbázis-kezelő szoftver. Az 5.0 előtti verzió használatát nem tartottam biztonságosnak, így azt elvetettem. A problémába Magyarország egyik legnagyobb szerver hosting szolgáltatójánál ütköztem. Ezt elég nagy problémának gondoltam ezt, hogy Érdemes legyen más adatbázis-kezelő rendszert keresni, a további ilyen problémák elkerülése végett, mivel egy felhasználó is beleütközhetett volna ebbe a problémába. Egy adatbáziskezelő rendszernél szerintem nagyon fontos a stabilitás és a kompatibilitás, és ebből a szempontból nem egy jó választás a MongoDB.

4.2.2 Time Series / Idősoros adatbázis

az idősoros adatbázisok minden beérkező adatot egy időbélyeggel egészítenek ki, és a beérkezési idő alapján indexelik az adatokat. Jellemzően az ilyen típusú adatbázisokat valamilyen rendszerességgel mért adatok időben történő változásainak követésére használják. Ha csak egy összesített decibel érték eltárolására lenne szükség, akkor ideális lehetne egy ilyen idősoros adatbázis használata. Egy idősoros adatbázist megnéztem részletesebben, az InfluxDB-t. Több olyan hasznos dolgot is nyújt ami az általánosabb az adatbázis-kezelő rendszer szintjén nem elérhető, és az adatbáziskezelő rendszert használó szoftverrel kell megvalósítani. Lehetőséget ad az adat időben változásának vizuális megjelenítésére a saját webes felületén, többféle időbeni aggregációs lehetőséget nyújt, mint például átlagolás és jobb sebességet ígér egy általános adatbázis-kezelő rendszernél. Amennyiben nem egy átfogó értéket akarunk figyelni, hanem egy 11-32 frekvenciatartományra lebontott tömböt, megváltozik a helyzet, mivel nem tud tömböket

kezelni a szoftver. Source Mivel nálam pont ez volt a helyzet, ezért újra kellett gondolnom.

4.2.3 SQL

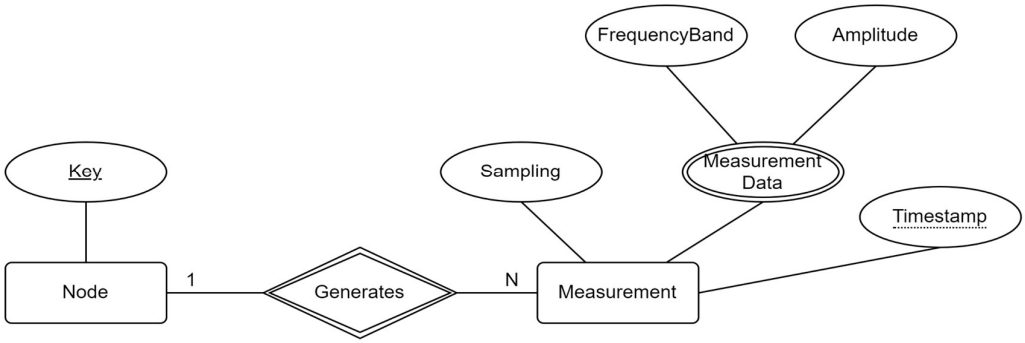
Az előző 2 lehetőség feltérképezését követően úgy döntöttem, hogy egy SQL alapú adatbázissal valósítom meg a rendszert, ezen belül is a MySQL [1] mellett döntöttem, mivel nyílt forráskódú, gyors, stabil és Linux kompatibilis.

4.2.4 Adatbázis felépítése

A következő adatokat kell eltárolni az adatbázisban:

- mérési eszköz azonosítója
- időbélyeg
- mintavételezési ráta
- mérési adatok.

Az azonosító és az időbélyeg szerepe triviális, a mérési adatok, mire az adatbázisba kerülnek már nominális frekvenciákból és a hozzá tartozó amplitúdókból állnak. A mintavételezési ráta, a mikrofon egy tulajdonsága, és onnantól hogy az előbbi mérési adatok létrejöttek minimális szerepet tölt be. Ennek az adatnak a megtartása csak diagnosztikai szerepet szolgál. Ha ezeket az adatokat szeretnénk rendszerezni, akkor az első elképzelés lehet talán, a mérőeszköz tulajdonságainak és a mérési adatok elkülönítése. Az ennek megfelelő modellt láthatjuk a 4.1. ábrán.

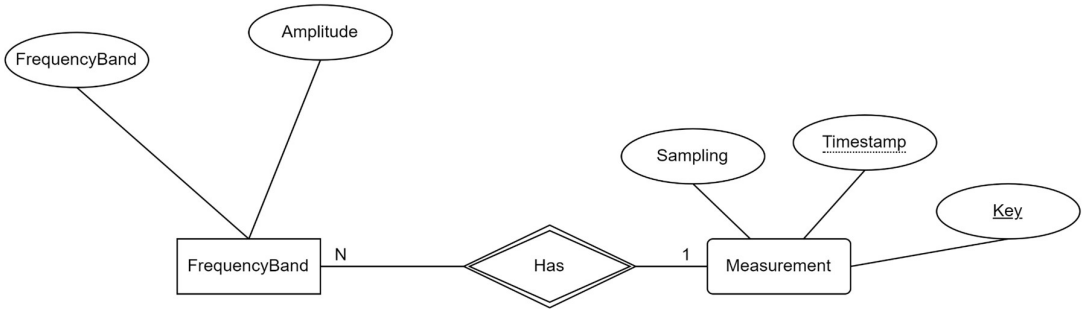


4.1. ábra EK-modell

A Nominális frekvenciákból és a hozzá tartozó amplitúdókból álló mérési adatokat egy külön táblába tettem, mivel a mintavételezési frekvencia függvényében változhat, hogy mi a legnagyobb frekvencia, ezáltal megváltoztatva a szükséges tulajdonságok számát, így nem lehetett volna ésszerűen kezelni az adatbázist másképpen.

Felmértem a lehetőségét, hogy a mintavételezési frekvencia egy külön táblában legyen. A tábla tartalmazná továbbá a mérőeszköz azonosítóját, amihez tartozik a frekvencia és az érvényességének a kezdete. Az adat küldésnél ugyanúgy elküldené a mérőeszköz, és az API ellenőrizné, hogy történt-e változás, és új rekordot hozna létre, különbség esetén. Lekérdezni úgy lehetne az adott méréshez tartozó frekvenciát, hogy az azonos azonosítójú eszközök közül ki kell választani azt a rekordot, aminek a kezdeti dátuma a mérést megelőzi, de azon belül a legkésőbbi.

Arra a következtetésre jutottam, hogy így nem működne gyorsabban a rendszer, mivel több lekérdezéssel és összehasonlítással járna ennek a bevezetése. Legjobb esetben memóriában lenne eltárolva minden eszköz legfrissebb frekvenciája és nem történne adatbázis lekérdezés minden új merésnél, de a mérések lekérdezésénél lassítaná a folyamatot. Az ennek megfelelő modellt láthatjuk a 4.2. ábrán. Az adatbázis tábláit szemlélteti a 4.3. és 4.4. ábra. A 4.5. ábra pedig a táblák közötti kapcsolatot mutatja be.



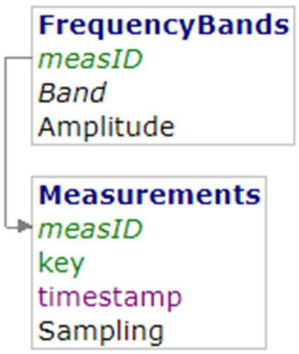
4.2. ábra módosított EK-modell

Column	Type	Comment
measID	varchar(255)	
key	longtext	
timestamp	datetime(6)	
Sampling	int	

4.3. ábra Measurement tábla

Column	Type	Comment
measID	varchar(255)	
Band	double	
Amplitude	double	

4.4. ábra FrequencyBands tábla



4.5. ábra táblák összeköttetése

4.2.5 Életciklus

Tesztek alapján az adatbázisban várható egy napi 5 GB/eszköz adatnövekedés, és 4 Mbps hálózati forgalom eszközönként. A terhelés folyamatos és egyenletes adatmennyiség időszaktól függetlenül, nem jelent kiugró értéket az sem, ha egyszerre van megállítva, elindítva több eszköz esetén sem. Jelenleg nincs beállítva elévülés, mivel ez minden környezetben változhat a felhasználói igényektől függően, és nem minden esetben elfogadható a régi adatok törlése.

4.3. A felhasználói felületek terve

A felhasználói felületnek 3 dolgot kell mindenképpen magába foglalni:

- Kezdőoldal: Szükséges egy összefoglaló nézet, ahol a felhasználó egy pillanat alatt ki tudja választani, hogy melyik eszköz adatait szeretné alaposabban megnézni.
- Grafikon: Kell egy olyan nézet, ahol a felhasználó részletesebben láthatja az élő adatokat egy adott eszközről grafikusan.
- Aggregálás: Valamilyen módon lehetővé kell tenni a felhasználónak, hogy vissza tudja nézni egy korábbi időpont zajszintjét.

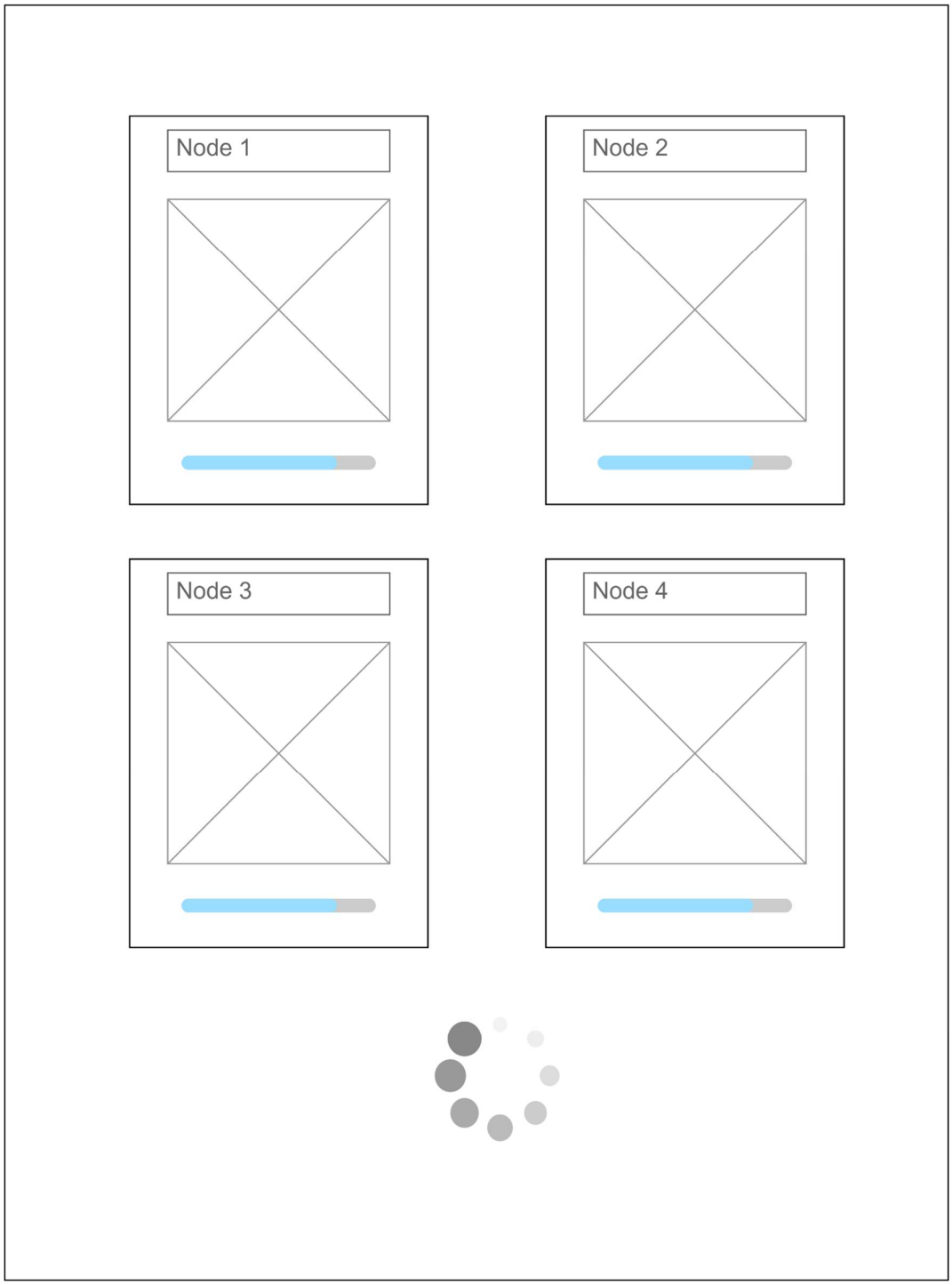
Ezt 3 dolgot úgy gondolom, hogy 2 oldalon lehet a leghatékonyabban megvalósítani, a kezdőoldal és a grafikus megjelenítést mindenképpen érdemes 2 külön lapra tenni, viszont az időpontra keresésnek elég egy menü ablak.

A weboldal kinézetét a Google Material Design formanyelv [1] alapján szeretném megcsinálni, mivel úgy gondolom, hogy a mögötte rejlő hosszú fejlődési folyamat, egy felhasználók számára könnyen használható szerkezetet eredményezett. A formatervezési nyelv széleskörű elterjedése és használata azt eredményezte, hogy a legtöbb felhasználó hozzászokott az erre épülő weboldalak viselkedéséhez. Ez tovább egyszerűsíti az interakciót az átlag felhasználó számára. A formatervezési nyelv megvalósítására a Vuetify keretrendszer mellett döntöttem, mivel ez egyik legnagyobb támogatással rendelkező keretrendszer, ami “Vue.js” környezetben valósítja meg a Google Material Design formanyelv specifikációit.

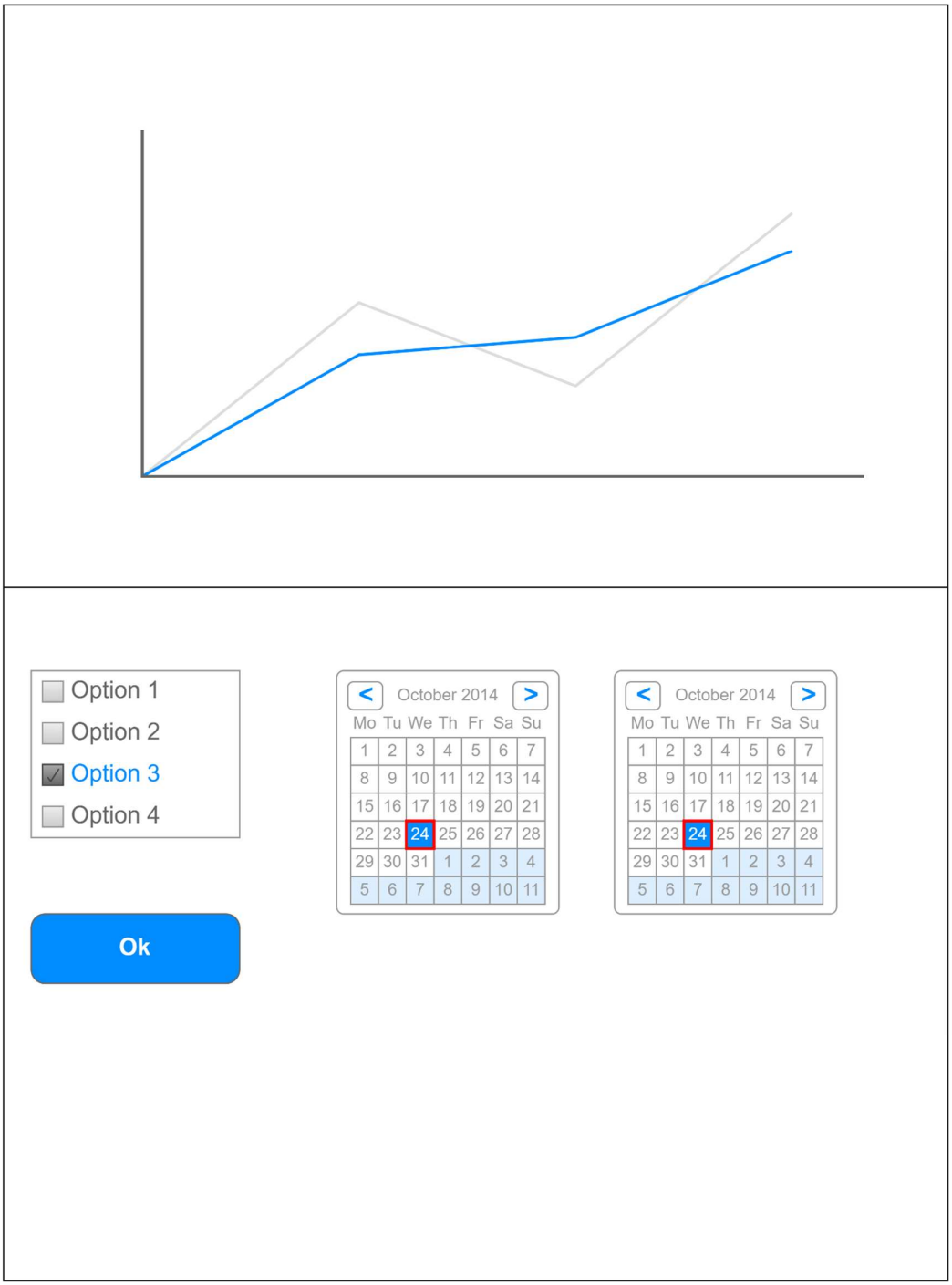
A kezdőoldal kinézetét ennek a dizájn nyelvnek az egyik komponense köré fogom felépíteni. Ez a komponens a kártya. A kártyák egyetlen témával kapcsolatos tartalmat és műveleteket jelenítenek meg. Több eszköz megjelenítésére jól alkalmazható ez a komponens. Egy ilyen kártyán az eszközzel kapcsolatos fontos információk lesznek csak rajta.

Ezeket a kártyák úgy lesznek elhelyezve az oldalon, hogy azok egymás mellett jelenjenek meg, ameddig kényelmesen elhelyezhetők egymás mellett, és ennél több eszköz esetén sorokba rendezve. Az oldal drótváza a 4.6. ábrán látható.

Az eszközök részletes nézete, a korábban említett két részből fog állni. A grafikon megvalósítására a “Chart.js” könyvtárat fogom alkalmazni, így nagyobb tervezést nem igényel ennek a kinézete. A menü egy olyan felületten lesz megvalósítva, ami eltüntethető, amikor nincs használatban, ezzel több teret adva a grafikonnak, és leegyszerűsítve az oldal kinézetét. A menüben lesz két dátumválasztó elem, amiknek az elhelyezése külön figyelmet igényel, a weboldal mobil eszközökre érvényesülő kinézeténél, az általuk elfoglalt hely miatt. A menü tartalmazni fog még egy lenyíló menüt az eszköz kiválasztására, és egy gombot az adatok megerősítésére. A gombot kihagyva megoldható lenne, hogy az oldal indítson egy lekérést, az aktuális paraméterekkel, amint változik valamelyik paraméter. Ebben a konfigurációban viszont, ez tovább rontana a helyzeten, ha a lekérések nem futnak le azonnal. A szerver számára felesleges terhelést jelentene, ezért döntöttem úgy, hogy erre a célra legyen egy gomb a menü felületén.



4.6. ábra A Főoldal drótváza



4.7. ábra A részletes nézet drótváza

4.4. API terve

Az API fontos részét képzi a rendszernek, a mérőeszközök és a kliensek számára is egy interfészként szolgál, biztosítva nekik az adatot egy olyan formában, ami megfelel a célnak. A felhasználási szempontok megvizsgálását követően arra jutottam, hogy érdemes az adatbázis sémától eltérő adatmodellt használni az adatfeltöltés és a megjelenítés esetében is.

Az adatfeltöltés esetében érdemes olyan sémát választani, amit a lehető legritkábban kell megváltoztatni, például ha mégse lenne elég a harmadoktávos kiosztás, és szeretnénk megváltoztatni a sávkiosztást, akkor ne kelljen átírni az összes használatban lévő eszközön a programot. A generált hálózati forgalom sem elhanyagolható emellett, habár ez kisebb problémát jelent egészen addig, amíg nem haladja meg a 100 Mbps hálózati forgalmat kliensenként. Elengedhetetlen az is, hogy a hálózati forgalomban csak frekvenciaadat jelenjen meg, ha a hanghullámok lennének elküldve és a szerveren történne a teljes transzformáció, akkor fennállna az esélye, hogy valaki lehallgatja a forgalmat, és ezzel rekonstruálhatná a hangot, amit az eszköz felvesz. A program ilyen módon való működése megnövelné a hálózati forgalmat is, de ez a probléma eltörpül a biztonsági kockázat okozta fenyegetés mellett. Minden tényezőt figyelembe véve, arra a következtetésre jutottam, hogy hosszútávon a legjobb módja az adatküldésnek, ha a transzformációt követően elküldi a kliens a szervernek a szükséges adatokat. Az ilyen módon elküldött adattömb nagyságrendileg 4000-6000 elemből fog állnia a mintavételezési frekvenciától függően. Szükséges a feldolgozáshoz a 2 frekvenciatartományú érték közötti különbség hertzben, hogy lehessen tudni a tömbben való indexe alapján a frekvenciát, amire az adat vonatkozik. A kliens el tud küldeni ezt az értéket, a mintavételezési frekvenciával együtt, így az itt használt adatmodell a következőképpen fog kinézni:

- eszközazonosító
- adattömb
- mintavételezés
- frekvenciakülönbség
- időbélyeg.

A megjelenítéshez ehhez képest teljes eltérő igények merülnek fel, itt a frekvenciát is szükséges átadni az amplitúdó mellett, méghozzá lehetőleg olyan szerkezettel érdemes ezt megtenni, ami a grafikon rajzoló könyvtárnak megfelel. Az én esetemben ez Vue-chartjs lesz, így megnéztem a minta diagramokat a szoftver dokumentációjában, és a mintának megfelelően neveztem el a grafikonon megjelenő adatokat. Ezzel megelőzve azt, hogy a böngészőben meg kelljen változtatni az adat formátumát kérésenként. Ez azt jelenti, hogy a frekvencia értékek egy "labels" tömbbe kell kerüljenek, az amplitúdókat pedig egy "datasets" objektum "data" tömbjébe, ahogy azt az 4.8. ábrán is láthatjuk.

chartConfig.ts

```ts
export const data = {
  labels: [
    'January',
    'February',
    'March',
    'April',
    'May',
    'June',
    'July',
    'August',
    'September',
    'October',
    'November',
    'December'
  ],
  datasets: [
    {
      label: 'Data One',
      backgroundColor: '#f87979',
      data: [40, 20, 12, 39, 10, 40, 39, 80, 40, 20, 12, 11]
    }
  ]
}

export const options = {
  responsive: true,
  maintainAspectRatio: false
}
```

4.8. ábra Vue.charts példa adat [https://vue-chartjs.org/examples/]

A grafikonhoz szükséges adatok mellet ez a lekérés tartalmaz egy "sum" mezőt, ami az egész spektrum átlag hangerejét tartalmazva, erre azért van szükség, hogy ha valahol egyszerűen akarjuk megmutatni a zajszintet, például a főoldalon, akkor azt könnyedén meg tudjuk tenni és nem kell egy plusz modellosztályt létrehozni, hanem mindkét cél esetében használható ugyanaz a séma. A plusz számítási kapacitás, amit ez igényel elhanyagolható, így nem indokolt a lekérés szétszedése.

4.4.1 POST /api/audio

Ezzel az API kéréssel lehet adatot bevinni a rendszerbe, a kliens eszközök használják. Az adat az API testrészben kerül átadásra.

4.4.2 GET /api/audio

Ezzel az API kéréssel lehet lekérni aggregált adatot, egy vagy több eszközről, egy meghatározott időtartományban keletkezett adatokat átlagolja az API, és adja vissza egy olyan formában, ami grafikon kirajzolására alkalmas.

Paraméterek:

- ids : \[string\]
- start : datetime-ISOstring
- end : datetime-ISOstring

4.4.3 GET /api/audio/{id}

Az URL-ben található eszközazonosító alapján a legutolsó beérkezett adatot adja vissza a kérés egy olyan formában, ami grafikon kirajzolására alkalmas.

5. KIVITELEZÉS

5.1 Kliens

A kliensprogram szerepét először egy Pythonban írt programmal szerettem volna kiváltani, mivel több olyan matematikai könyvtár is rendelkezésre állt, amivel könnyen elvégezhető volt a szükséges transzformáció és a grafikonra rajzolás, az eredmény vizualizálásához.

A programnak ezt a verzióját azonban hamar elhagytam, és csak prototípusként volt használva a hangfeldolgozás megismeréséhez. A program elhagyásának hátterében az állt, hogy elkezdett nehezen tesztelhető lenni, mivel nem állt rendelkezésemre egy hibakeresési funkciókkal rendelkező fejlesztői környezet, és emellett több olyan

adatosztály és hangfeldolgozó függvény volt a szerveroldali kódban, amit fel tudtam használni az újraírást követően.

A Kliens program egy C#-ban írt .NET Core 6 [3] parancssoros alkalmazás, ami egy YAML konfigurációs fájlból olvassa be a rögzítőeszközre vonatkozó paramétereket.

5.1.3 Hangrögzítés

Sokáig kerestem a megfelelő könyvárat, ami lehető teszi, hogy olyan formátumban rögzítsem a hangadatokat, amiből ki lehet nyerni a megfelelő, frekvenciatartományra vonatkozó adatokat. A legtöbb könyvtár fizetős vagy hiányzik belőle a platformfüggetlenség, ami elengedhetetlen volt ahhoz, hogy a használt Linux környezetben működjön a szoftver.

A fejlesztés kezdetleges fázisában megkerülve a problémát a feldolgozás előtt a hangadat egy .WAV formátumú fájlba került, mivel fájl formátum adat része PCM adatként van tárolva, bármiféle tömörítéstől mentesen. Világos volt azonban, hogy ez csak egy ideiglenes állapot lehet. A fejlesztésnek ebben fázisában továbbá egy python programot használtam prototípusként, mivel ezt volt a legegyszerűbb opció a fejlesztésnek ebben a fázisában, és ehhez tudtam a legtöbb segítséget találni az interneten. Onnantól kezdve, hogy egy API kérést kellett küldeni a kliensből, jobb döntéssé vált a .NET kód használata, így újra tudtam használni a modellosztályt, ami eddig az API része volt, és néhány funkciót a DataConverter osztályból.

Annak ellenére, hogy itt már nem egy fájlba lett mentve a felvett hang, kezdetben itt is problémákba ütköztem, mert azt feltételeztem, hogy a hang rögzítésére használt könyvtár kompatibilis lesz az eszközön futó Linux környezettel. A feltételezésem hátterében az állt, hogy csak olyan forrást találtam, ami szerint a könyvtár kompatibilis Linux környezettel, ennek a forrásnak a megbízhatósága kétes volt, és azóta el is tűnt, azonban nem találtam olyan forrást, ami cáfolta volna ezt az állítást. A ténnyel, hogy könyvtár által támogatott hardver-illesztőprogramok csak windows platform alatt léteznek, és ezáltal nem működőképes a könyvtár egy Linux környezetben, akkor szembesültem, amikor a működőképes programot, miután a fejlesztésre használt Windows 10-es gépemen tesztelve megírtam, megpróbáltam feltenni a célgépre.

Ezután találtam rá a Portaudio könyvtárra, ez egy nyílt forráskódú platformfüggetlen könyvtárra, ami támogatja a hangrögzítés kezelését Linuxon is. A könyvtár használat először ugyan egy plusz nehézséget jelentett, mivel nem egy .NET könyvtárral volt dolgom, hanem egy C-ben írt könyvtárral, amit egy dinamikus csatolású könyvtár formájában tudtam használni, azonban a probléma megoldásához így is közelebb vitt.

Először nem tudtam, hogy egy ilyen könyvtárat hogyan tudok alkalmazni, de rátaláltam a PortAudioSharp-ra, egy wrapper könyvtárra, amely kiküszöbölte a közvetlen P/Invoke hívások szükségességét. Ennek a könyvtárnak a használata eltért attól, amit másik könyvtárak használatakor megszokhattam. Nem szabadított meg teljesen a natív kód komplikált integrációjától, azonban más hasonló könyvtár nem állt rendelkezésre. A visszahívási függvényben kaptam egy-egy mutatót, ami a bejövő és a kimenő hangra mutat, és egy számot, ami a minták darabszámát tartalmazza, illetve egy időinfót, státuszjelző zászlókat és egy mutatót a felhasználói adatokra. A PortAudio dokumentációját [4] megvizsgálását követően megtudtam, hogy a bemeneti eszköz paraméterei között tudom definiálni azt, hogy milyen értéke legyen a mintáknak, így már csak be kellett állítani ezt a paramétert, és a .NET Marshal osztályával be kellett olvasnom a bemeneti mutatón található értékeket, a paraméterként beállított adattípusnak megfelelően.

5.1.4 Hangfeldolgozó könyvtár

A felvett hang feldolgozására az FFTSharp könyvtárat használtam, amire egy olyan weboldalon találtam rá, ahol a könyvtár szerzője mutatja be annak a működését, egy projekten keresztül. [2] A könyvtár működését is ezen az oldalon keresztül értettem meg. A könyvtár megvalósítja a Zajméréshez szükséges Gyors Fourier-transzformációt, továbbá az adatok előkészítésének az érdekében, lehetővé teszi az ablakfüggvények érvényesítését és az adatsor nullákkal kibővítését.

5.1.5 Működés

Működés során a kliensoldali program egy mikrofon segítségével veszi fel a hangot folyamatosan, és amikor megtelt egy 250 milliszekundumnak megfelelő méretű buffer, akkor feldolgozza az összegyűlt adatokat a következőképpen:

Az első az adat letisztítása, a program egy Hamming ablakfüggvényt alkalmaz az adatsoron, ezzel csökkentve a mérési zajt.

Ezután az adatsor ki van párnázva nullákkal a legközelebbi 2 hatványára. Ez azért fontos, mert a transzformáció, amit használ a program, így a leghatékonyabb, és a könyvtár ami a transzformációt végrehajtja megköveteli ezt. Amennyiben ennek nem felel meg az átadott adatsor, hibát dob a program.

A következő lépés a gyors Fourier-transzformáció, ami átalakítja a hangfelvételt frekvencia spektrummá. Ennek az adattömbnek az értéke megegyezik a bufferméretnél nem nagyobb legközelebbi kettő hatványával (,azaz a korábban felfele kerekített buffer méretének a fele). Az itt használt függvény decibel értéket ad vissza.

A program egy statikus "Sender" osztályt használ az adat elküldésére egy publikus "SendData" metóduson keresztül, aminek a bemeneti paraméterei a következőképpen néznek ki:

- key : string
- data : double[]?
- sampling : int
- freq : double
- timestamp : DateTime.

Ezt a modellosztályt JSON formátumban szérializáljuk, mielőtt továbbítanánk UTF-8 kódolással az API felé.

5.1.5 Naplózás

A program a Serilog könyvtárat használja a naplózás egyszerű és hatékony kezeléséhez. Jelenleg az alkalmazás az alapértelmezett kimenetre naplóz, de igény szerint könnyedén beállítható egy másik naplózási cél is, akár fájlba, vagy akár egy Syslog protokollt használó szerverre. A program induláskor kiírja a felismert hangrögzítő eszközök számát, az alapértelmezett eszköz sorszámát és mintavételezési frekvenciáját. Ezután kiírja a

kiválasztott eszköz sorszámát és mintavételezési frekvenciáját, és a program által használt mintavételezési frekvenciát, ami eltérhet eszközétől.

Amennyiben az eszköz meghajtójában változás történik, vagy az eszköz operáció rendszerén átállítja ezt valaki, akkor jobb ha ez a programban nem változik meg, ezért nem állítja be a program ezt az értéket önállóan. Amennyiben egy nagyobb változás történik, és ki van cserélve a hangkártya amin keresztül történik a zajmérés, akkor a naplózás emlékeztethetőként lehet jelen a felhasználó számára, hogy nem lett átállítva ez az érték a beállításokban. A tényleges mérés indulásakor az első 5 mérés eredményét feldolgozza ahhoz hasonlóan, ahogy az a szerveren is fel lenne dolgozva. A feldolgozott eredményt egy olyan formátumban jeleníti meg, hogy az egymás alatt megjelenő üzenetek vizuálisan egy táblázatot alkossanak. Ezt az 5.1. ábrán lehet látni.

```
09:57:39 INF  found 3 Devices
09:57:39 INF  Default Device #2 (Sampling: 44100)
09:57:39 INF  Using #2 (Sampling: 44100)
09:57:39 INF  Sampling Rate set to: 44100
09:57:40 INF  Band                    Value
09:57:40 INF  ----------------------------------------
09:57:40 INF     15.625000      -33.572753
09:57:40 INF     19.686266      -69.423627
09:57:40 INF     24.803141      -77.487548
09:57:40 INF     31.250000      -75.729734
09:57:40 INF     39.372533      -75.770058
09:57:40 INF     49.606283      -49.049775
09:57:40 INF     62.500000      -51.659073
09:57:40 INF     78.745066      -74.599361
09:57:40 INF     99.212566      -76.446478
09:57:40 INF    125.000000      -76.147924
09:57:40 INF    157.490131      -61.805755
09:57:40 INF    198.425131      -78.037492
09:57:40 INF    250.000000      -70.547826
09:57:40 INF    314.980262      -80.343656
09:57:40 INF    396.850263      -78.854064
09:57:40 INF    500.000000      -80.608179
09:57:40 INF    629.960525      -82.441397
09:57:40 INF    793.700526      -82.064584
09:57:40 INF   1000.000000      -82.097288
09:57:40 INF   1259.921050      -85.192993
09:57:40 INF   1587.401052      -86.754151
09:57:40 INF   2000.000000      -87.517322
09:57:40 INF   2519.842100      -88.531497
09:57:40 INF   3174.802104      -89.425800
09:57:40 INF   4000.000000      -90.096601
09:57:40 INF   5039.684199      -90.492857
09:57:40 INF   6349.604207      -90.718535
09:57:40 INF   7999.999999      -90.265284
09:57:40 INF  10079.368398      -90.568512
09:57:40 INF  12699.208414      -89.933201
09:57:40 INF  15999.999998      -89.541810
09:57:40 INF  20158.736796      -89.056527
```

5.1.ábra A kliens naplózási kiemente

5.1.6 Eszköz

A mérésre használt eszköz fontos, hogy ne foglaljon sok helyet, alacsony áramfogyasztása legyen, halk legyen, tudjon a hálózatra csatlakozni és lehessen mikrofont csatlakoztatni rá. A két fő jelölt erre a célra egy Arduino és egy Raspberry Pi volt. A kettő közül az Arduino egy kicsit kompaktabb volt és halkabb, viszont problémásabb lett volna hálózatra tenni, vagy mikrofont kötni rá, mivel egy olyan mikrofont vagy hangkártyát kellett volna találni, amit rá lehet forrasztani. Lehet hozzá

kapni zajmérő szenzort, de ez csak egy összesített értéket ad vissza. A Raspberry pi esetében egyszerűbb a helyzet, mivel a főbb modellek esetében van vezetékes és vezeték nélküli hálózati csatlakozási lehetőség is, és van rajta USB port egy külső hangkártyának. Ezért a Kliens program futtatásához egy Raspberry Pi 4 Model B eszköz mellett döntöttem.

5.2 API

Az API szerver létrehozásához egy .NET Core 6 Web API projektet vettem alapul, majd létrehoztam egy új Kontrollert AudioController néven. Ahhoz, hogy kéréseket hozzak létre az új kontrollerben, előtte létre kellett hoznom a modellosztályokat. Először is ahhoz, hogy a kliensprogramból tudjak küldeni adatot, az ehhez tartozó "Datapoint" modellosztályt kellet létrehoznom. az osztályt a következő tulajdonságokkal hoztam létre:

- key
- data
- sampling
- freq
- timestamp.

```C#
public class Datapoint
{
        [Required]
        public string key { get; set; }
        [Required]
        public double[]? data { get; set; }
        [Required]
        public int Sampling { get; set; }
        [Required]
        public double freq { get; set; }
        [Required]
        public DateTime timestamp { get; set; }
}
```

5.2. ábra Datapoint osztály

A következő az adat adatbázisban tárolásához szükséges modellosztályok, az adatbázis kezeléséhez a .NET Entity Framework-öt, egy népszerű .NET keretrendszert használtam. [6] Ez a keretrendszer lehetővé teszi, hogy egy objektum orientált program modell osztályaival definiáljam az adatbázisom szerkezetét, és ennek a segítségével lehet létrehozni, és lekérdezni az adatokat. Ezt lehetővé tette, hogy az adathozzáférés és adatbázis-műveletek helyett az üzleti logikára koncentráljak.

A következő modellosztályokat használtam az adatbázis létrehozásához és használatához.

Measurement

- key: ez lesz a korábban említett Mérési eszköz azonosítója, érdemes figyelembe venni, hogy a név ellenére nem ez az elsődleges kulcs az adatbázisban, mivel ez csak az eszközt azonosítja, a mérést nem.
- timestamp: a mérési adatok létrejöttekor lementett időbélyeg
- Sampling: a használt Mintavételezési ráta
- measID: ez a tulajdonság egy automatikusan generált kulcs, ami egyértelműen beazonosítja a méréseket.
- Bands: Navigációs szerepet szolgál, például hogyha van egy Measurement típusú measurement nevű változónk, amit az adatbázisból kértünk le, és a mérési adatokhoz szeretnénk hozzáférni, akkor azt könnyedén megtehetjük a `measurement.Bands` tulajdonságon keresztül.

```C#
public class Measurement
{

    public string key { get; set; }
    public DateTime timestamp { get; set; }
    public int Sampling { get; set; }
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public string measID { get; set; }
    public virtual List<FrequencyBand> Bands { get; } = new();

}
```

5.3. ábra Measurement osztály

A virtual kulcsszó lehetővé teszi a Programnak, hogy "Lazy Loading"-ot használjon, így csak akkor kéri le a másik táblából az adatokat, amikor hozzáférünk. [msft](https://learn.microsoft.com/hu-hu/ef/ef6/modeling/code-first/workflows/new-database)

FrequencyBand

- measID: az idegen kulcs az előző modellosztályból
- Band: a frekvenciatartomány nominális értéke
- Amplitude: a frekvenciatartomány amplitúdója
- Measurement: navigációs tulajdonság hasonlóan a `measurement.Bands`-hez

```C#
public class FrequencyBand
{
    public string measID { get; set; }
    public double Band { get;set; }
    public double Amplitude { get;set; }
    public virtual Measurement Measurement { get; set; }
}
```

5.4. ábra FrequencyBand osztály

Az Entity Framework használatakor, hogy ténylegesen adatbázis műveleteket lehessen végrehajtani, szükség van egy DBContext osztályra. A DBContext felel azért, hogy a modell osztályok és az adatbázis táblák között megfelelően működjön a leképezés. Egy ilyen DBContext osztályt hoztam létre én is, ebben meghatároztam, hogy melyik modellosztályokat szeretném adatbázistáblaként használni, és definiáltam, a közöttük létrejövő kapcsolatot. Az osztály felépítése az 5.5. ábrán látható.

```
9 references
public class AudioContext : DbContext
{
    4 references
    public DbSet<Measurement> Measurements { get; set; }
    1 reference
    public DbSet<FrequencyBand> FrequencyBands { get; set; }
    0 references
    public AudioContext(DbContextOptions<AudioContext> options) :base(options) { }

    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Measurement>()
            .HasMany(e => e.Bands)
            .WithOne(e => e.Measurement)
            .HasForeignKey(e => e.measID)
            .HasPrincipalKey(e => e.measID);
        modelBuilder.Entity<FrequencyBand>()
            .HasKey(e => new { e.measID, e.Band });
        ///base.OnModelCreating(modelBuilder);
    }
}
```

5.5. ábra AudioContext osztály

Annak érdekében, hogy minél egyszerűbb legyen a kliens program által használt DataPoint, és az adatbázishoz használt Measurement osztály közötti konverzió, a DataPoint osztályt kiegészítettem egy ToMeasurement metódussal, ami létrehoz egy Measurement objektumot, az eredeti objektumnak megfelelő adatokkal, ehhez a konverzióhoz szükséges a Datapoint objektum adatait frekvenciasávokba aggregálni. A program a DataConverter osztály függvényeinek a segítségével teszi ezt meg. A függvény felépítése az 5.6. ábrán látható.

```C#
public Measurement ToMeasurement()
{
    Measurement measurement = new Measurement();
    measurement.key = this.key;
    measurement.timestamp = this.timestamp;
    measurement.Sampling = this.Sampling;
    var values = DataConverter.GetTriOctaveDecibels(this).ToList();
    var keys = DataConverter.GetBands(true).ToList();

    if (values.Count != keys.Count) { throw new Exception("the number of band
keys and values do no match"); }
    else
    {
        for ( var i = 0; i < values.Count; i++ ) {
            FrequencyBand f = new FrequencyBand();
            f.Amplitude = values[i];
            f.Band = keys[i];

            measurement.Bands.Add(f);
        }
    }

    return measurement;
}
```

5.6. ábra Datapoint.ToMeasurement függvény

5.2.1 POST /api/audio

Ezután létrehoztam lekéréshez szükséges metódusokat. a POST metódus, amit a kliensprogram használ nagyon egyszerűen működik. A DataPoint objektumot átkonvertálja a korábban említett ToMeasurement metódussal, majd hozzáadja az adatbázishoz az Entity Framework segítségével, mielőtt visszaadná az eredményt.

```
[HttpPost]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
0 references
public async Task<IActionResult> PostData(Datapoint datapoint)
{
    db.Measurements.Add(datapoint.ToMeasurement());
    db.SaveChanges();

    return await Task.FromResult(CreatedAtAction(nameof(Get), new { id = datapoint.key }, datapoint));
}
```

5.7. ábra POST metódus

A megjelenítéshez szükséges lekérések már nem ilyen egyszerűek, mivel itt meg kell határozni, hogy mi az az adat amit le szeretnénk kérni az adatbázisból, a lekérésben található adatok alapján.

5.2.2 GET /api/audio/{id}

Ez az a lekérés az, ami az utolsó állapotát kéri le egy eszköznek, ezt egy LINQ kifejezéssel oldottam meg, ami kiválasztja az adott eszközhöz tartozó mérési eredményeket, és ezek közül kiválasztja a legújabb időbélyeggel rendelkező mérést. [7] A LINQ kifejezés felépítése az 5.8. ábrán látható.

```
[HttpGet("{id}")]
1 reference
public ViewPoint Get(string id)
{


    var measurement = (from m in db.Measurements
                       where m.key == id
                       orderby m.timestamp descending
                       select m).FirstOrDefault();
    if (measurement == null)
    {
        throw new ArgumentException("ID {id} not found", id);
    }
    var bands = from f in db.FrequencyBands
                where f.measID == measurement.measID
                select new
                {
                    keys = f.Band,
                    values = f.Amplitude,

                };
```

5.8. ábra GET metódus

Ezt követően kiszedi a frekvenciákat és az amplitúdókat egy-egy tömbbe, kiszámolja a mérés összesített zajszintjét, és visszaadja ezt a három értéket.

5.2.3 GET /api/audio

Ezzel a lekéréssel lehet minden aggregált adatot lekérni, az ehhez szükséges paramétereket a fejlécben várja a szerver, ezt Attribútumokkal tudtam meghatározni a szerver kódjában. Erre azért volt szükség mert a szoftver nem támogatja, hogy a URL-ben legyen a paraméter, amennyiben az egy tömb, és az egységesség érdekében minden paramétert a fejlécbe tettem át. Ezeket az attribútumokat az 5.9. ábrán láthatjuk a gyakorlatban.

```
[HttpGet]
1 reference
public ViewPoint Get
    (
    [FromHeader(Name = "ids")] List<string> ids,
    [FromHeader] DateTime start,
    [FromHeader] DateTime end
    )
```

5.9. ábra FromHeader attribútum

Ennél a lekérésnél a LINQ kifejezés annyiban módosul előző lekéréshez képest, hogy egy eszköz azonosító helyett egy lista alapján, minden olyan mérést ki kell választani, ami a lista valamelyik tagjához tartozik, továbbá a legutolsó dátumú mérés helyett minden olyan mérésre szükség van, ami a két megadott időpont között van. Ha ez megtörtént, akkor már csak ki kell számolni átlagot, és az előző lekérés mintájára visszaadni az adatot. Az átlag számításánál figyelni kell azonban arra, hogy ez egy logaritmikus decibel érték, ezért előtte vissza kell alakítani lineáris skálázású adattá. Az átlagolást követően a megkapott értéket ismét logaritmikus decibel adattá kell alakítani.

5.2.4 CORS

Miután az API felkerült a szerverre azt tapasztaltam, hogy lekérések igazából működnek, de ha egy weboldalon keresztül próbálom használni azokat, akkor nem jelenik nem ez az adat a weboldalon. Egy kisebb hibakeresés után rájöttem hogy ez a CORS (Cross-Origin Resource Sharing) által lett letiltva. Ez egy biztonsági protokoll, amit a modern böngészők figyelnek. Az API lekérés válaszában kell legyenek bizonyos fejlécek, amik meghatározzák, hogy milyen weboldalak használhatják ezt a lekérést. Amikor egy weboldal tartalmaz egy API lekérést, akkor a böngésző küld egy "preflight" kérést, és az erre a kérésre kapott válaszban lévő fejlécek alapján dönti el, hogy elküldi-e a tényleges

lekérést. [3] Annak érdekében, hogy együtt tudjon működni az API szerver és a weboldal, beállítottam a CORS beállításokat az API kódjában. Ezt egy szolgáltatásként tudtam hozzáadni az alkalmazáshoz a builderen keresztül, amit az újonnan létrehozott projekt is alkalmazott, a Program osztályban.

5.2.5 Mérési kalibráció

Ahhoz, hogy a mérési eredmények ne csak az eszköz korábbi méréseihez lehessen viszonyítani, szükséges egy viszonyítási alap. Erre a célra, azaz akusztikai méréseknél a 0 dB pontra, általánosságban elterjedt a 20 μPa légnyomás használata. [4] Mivel ezt a környezetet nem tudom előallítani az eszköz kalibrációjához, ezért egy másik zajszint mérő műszert tettem mellé, és a két eszköz különbsége alapján állítottam be a kalibrációt.

A kalibrálást az API végzi az adatbázisból kiolvasást követően, mivel így a mindenkori mérésekre lehet érvényesíteni. Az API szerveren központilag megoldható a kalibráció konfigurálása.

5.3 Web

5.3.1 Vue.js

A webes frontend fejlesztéséhez Vue.js keretrendszert használtam. A Vue.js egy JavaScript keretrendszer, ami megkönnyíti a felhasználói felületek létrehozását. [9] A fejlesztési folyamat .vue végződésű fájlokkal történik egy HTML-en alapuló, kiegészített szintaxissal. A keretrendszer középpontjában állnak a komponensek, amik felhasználói felület önálló részei, a saját tulajdonságaikkal, eseményeivel és belső logikájukkal. A definiálásukat követően ezek a komponensek létező html címkékhez hasonlóan használhatóak.

5.3.2 Vue-router

A Vue-router a Vue.js egy hivatalos kiegészítője, ami lehetővé teszi, hogy az weboldal lapjai között navigáljunk, annak az újratöltése nélkül. Ennek is egy olyan konfigurációját használtam, ami automatikusan generálja az elérhető útvonalakat. Mindössze annyi a dolgunk a használatakor, hogy egy előre létrehozott "pages" mappán belül hozunk létre

komponenseket, például ha létrehozunk egy "index.vue" fájlt, és definiáljuk benne a főoldalon látni kívánt tartalmat, akkor ezt később el fogjuk érni az /index elérési útvonal segítségével.

5.3.3 Vuetify

A Vuetify egy nyílt forráskódú, felhasználói interfész elemeket tartalmazó könyvtár. Előre legyártott, testre szabható komponenseket tesz elérhetővé, a fejlesztési folyamat felgyorsítása érdekében. A komponensek mellett a weboldal kinézetének formálására olyan osztályokat is biztosít, amelyeket hozzá lehet adni komponensekhez vagy natív html címkékhez, hogy a weboldal kinézetét minél könnyebben testre lehessen szabni.

5.3.4 Felépítés

A felhasználói felület kialakításakor 2 oldalt hoztam létre: Az egyik a főoldal, ahol láthatjuk a különböző eszközöket kilistázva egy képpel, ami lehet az eszköz helye egy térképen, az általa mért terület vagy bármi, ami lehetővé teszi a gyors felismerést, valamint az utolsó mért hangerővel.

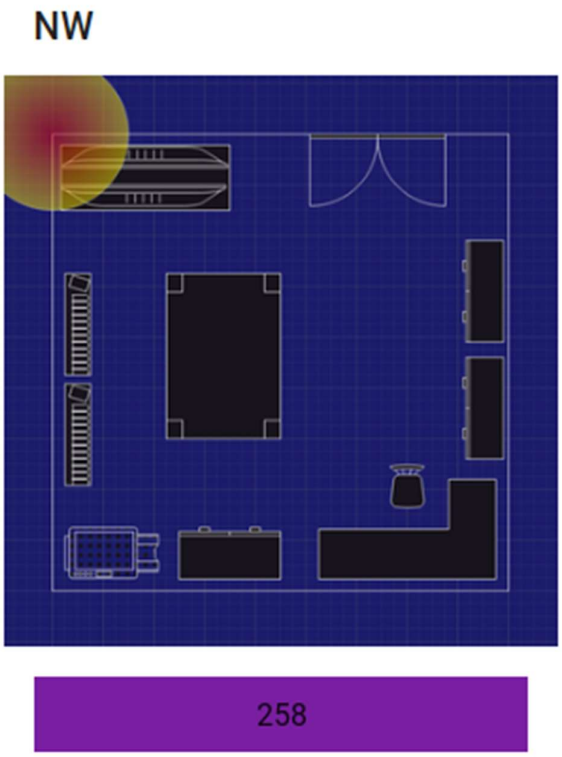
Ezáltal azonnal láthatjuk, ha van olyan eszköz, ami figyelmet igényel.

A másik oldalon ahol egy adott eszköz által mért zajszintet láthatjuk, frekvencia tartományokra bontva egy grafikonon, ha időpontra szeretnénk keresni, akkor azt is itt tehetjük meg, egy menü segítségével.

5.4.5 Komponensek

NodeCard

Ez a komponens a főoldalon található eszközlista egy-egy elemének a megjelenítését határozza meg.



5.10. ábra NodeCard komponens

Tulajdonságai:

- title
- uri
- id.

A "title" tulajdonság segítségével lehet beállítani az eszköz nevét, ami kártya legtetején található.

A "uri" tulajdonsággal meghatározható az adott eszközhöz tartozó API elérési útvonal.

Az "id" tulajdonság az eszköz azonosítóját tartalmazza. Ennek az a szerepe, hogy a weboldal a megfelelő oldalra tudjon minket irányítani, ha rákattintunk a kártyára.

A komponens tartalmaz továbbá két belső tulajdonságot, amik a lekérdezett adatok alapján kapnak értéket. Az egyik a hangerő alapján beállított Zajmutató csíknak a színét állítja be, a másik pedig Azt mondja meg, hogy mennyire legyen tele a csík a zajszint alapján.

A komponens ezen kívül meghatározza, hogy a megjelenítendő kép helyét, amit a szülő komponens a címkén belül elhelyez, hova kerüljön. Ezt egy úgynevezett “slot” segítségével határozza meg a komponens. A szülő elhelyez egy "image" azonosítójú template címkét és, amit a szülő komponens a címkén belülre tesz, azt a megfelelő helyre beilleszti a gyermek komponens. Ennek a használata az 5.11. ábrán látható.

```vue
<NodeCard class="ma-3 pa-3" v-for="d in nodes" :title="d.title"
:uri="uribase+d.id" :id="d.id">
<template #image>
<v-img :src="d.imgsrc"></v-img>
</template>
</NodeCard>
```

5.11 ábra

Graph

A zajszint frekvenciákra bontott grafikus megjelenéséért felelős komponens.

Tulajdonságai**:**

- requestoptions
- requesturi.

A komponens a saját megjelenítési adatainak lekérdezéséhez indít API lekéréseket, azonban ennek a kérésnek a tulajdonságai függenek egy másik komponens állapotától, ezért szükséges, hogy kívülről módosítani lehessen a kérés elérési útvonalát és a kérés fejlécét. Annak érdekében, hogy a komponens által indított API lekérdezés kívülről módosítható legyen, tulajdonságként kell definiálni a kéréshez tartozó adatokat. Az elérési útvonalat csak akkor szükséges módosítani, amikor a 2 API végpont között történik váltás azaz, amikor az adott eszköz legfrissebb adata helyett egy aggregált nézetre

szeretnénk váltani. A kérés fejlécét akkor szükséges megváltoztatni, amikor az aggregált nézet paramétereit szeretné a felhasználó finomítani, mivel ezeket a paramétereket az API a fejlécben keresi.

A komponens periodikusan megismétli a lekérdezést, ha nem történik külső behatás akkor is, továbbá tartalmaz egy gombot, amivel manuálisan lehet kezdeményezni egy frissítést.

A grafikon megrajzolására egy nyílt forráskódú "Chart.js" nevű grafikon rajzoló könyvtárat használtam. A grafikon kinézetének definiálását követően, a megfelelő szerkezetű adat objektum beinjektálásával megjeleníti az objektum által tartalmazott adatot. Ezen a könyvtáron kívül használtam még egy "vue-chartjs" nevű könyvtárat is, ami a Chart.js és Vue.js közötti integrációt segíti elő.

Selector

A korábbi mérések, és az aggregált adatok lekéréséhez használható menüt tartalmazó komponens.

A komponens egy "graphuri" nevű tulajdonságot definiál, amin keresztül meg lehet adni neki, hogy mi az eszköz azonosítója, amire a felhasználó rákattintott, így be tudja állítani alapértelmezettnek a eszközök listájában.

A komponens definiál egy eseményt, amit a "Submit" gomb lenyomását követően vált ki.

A Menü elemei különböző tárolókba vannak helyezve, a reszponzív megjelenés érdekében. Megfigyelhető 3 ilyen tároló. az első tartalmazza az eszközök kiválasztására szolgáló listát, egy kapcsolót, aminek a segítségével megmondhatja a felhasználó, hogy mennyire pontosan szeretné megadni az időpontokat, és ezen kívül még az előbb említett "Submit" gombot tartalmazza az első tároló. A kapcsolóval választható, hogy kizárólag a napokat szeretnénk megadni, vagy a pontos időpontokat. Amennyiben az előbbit választjuk, a idő beállításhoz szükséges mezőket automatikusan elrejti a program. A második tároló tartalmaz egy Dátum kiválasztására szolgáló vuetify komponenst, és egy időválasztó mezőt. Ezekkel a lekérni kívánt szakasz elejét tudjuk meghatározni. A harmadik tárolónak a felépítése hasonló a másodikéhoz, annyi különbséggel, hogy ez a lekérni kívánt szakasz végére vonatkozik.

5.4.6 Lapok

A weboldal két oldalt különböztet meg, és annak ellenére, hogy ezek is komponensként funkcionálnak, azt gondolom, hogy érdemes elkülöníteni őket az előbb említett komponensektől.

Audio

Ez az oldal megjeleníti a "Graph" és "Selector" komponenseket, a Selector eseményére feliratkozik és meghíváskor átadja a Grafikonnak a frissült adatokat.

Index

A főoldal egy JSON fájlból olvassa ki egy listát, ami tartalmaz minden eszközhöz egy azonosítót, Címet, és az eszközhöz tartozó kép elérési útvonalát. A beolvasott listából létrehoz egy kártyát minden eszközhöz, és megjeleníti azokat.

5.4 DataConverter

A DataConverter osztály egy statikus funkciógyűjtemény, ami a formátumok közötti átváltásért és ehhez kapcsolódó kiegészítő funkciókért felel. Főként az API szerver használja, de egy részét a kliens program is használja naplózáshoz.

Az osztály legnagyobb feladata a frekvencia adatok sávokba csoportosítása. Ahhoz, hogy ezt a leghatékonyabban tehesse meg a program, futáskor legenerálja a sávok közötti elválasztó értékeket, az oktáv szerinti felosztás és a harmadoktáv szerinti felosztáshoz is. A határértékek legeneráláshoz először a középértékek kellenek. A két felosztás legenerálása nagyon hasonló egymáshoz, csak más értékekkel behelyettesítve. Az oktáv esetében kettővel kell szorozni a frekvenciát, hogy megkapjuk a következő oktávot, a határértékek megkapásához pedig kettőnek a négyzetgyökével kell szorozni a középértéket a felső határhoz, és osztani az alsó határhoz. A harmadoktáv esetében kettőnek a köbgyöke lesz a két középérték aránya, a határértékek kiszámításához szükséges szám pedig kettőnek a hatodik gyöke lesz. A program először létrehozza a középértékek listáját. Ehhez annyit csinál, hogy megszorozza a kezdőértéket, ami jelen esetben 15.625, a megfelelő értékkel egészen addig, amíg el nem éri a felső határt. A

programban ez a felső határ 21000, így a 32. harmadoktávot még kiírja, aminek a középértéke 20158.737 Hz. A megkapott középértékekből ezután a program kiszámítja a elválasztó értékeket, mivel az N-edik oktáv felső határa és az N+1-edik oktáv alsó határa szükségszerűen egybevág, ezért elég mindegyik középértékhez tartozó felső határt kiszámolni. A kiszámolt értékeket a program ezután eltárolja a memóriában.

Az osztály tartalmaz egy nyilvános függvényt, ami a egy decibel számsor összeadására használható. A függvény először átalakítja az értékeket lineáris skálájú értékre, majd azokat összeadja és végül visszaalakítja logaritmikus decibel értékre. Ez a függvény fontos az adatsor sávokra bontásához is, és ahhoz is, hogy egy összesített értéket lehessen kiszámolni a zajszintről.

Az osztály elérhetővé tesz egy-egy függvényt az oktávokra, és a harmadoktávokra bontáshoz. Mindkét metódus ugyanazt a privát metódust hívja meg futáskor, és átadja a megfelelő eltárolt határértékeket bemeneti paraméterként, a mérési adatokat tartalmazó Datapoint objektum mellett. A meghívott privát metódus egy LINQ kifejezés segítségével kiválasztja az adattömbnek egy szakaszát, ehhez a két határértéket használja és az adat frekvenciafelbontását, amit a Datapoint osztály tartalmaz. Az említett adatok segítségével könnyedén kiszámolható, hogy milyen két index között között található a frekvenciasávhoz tartozó adat a mérési adatban. A függvény szerkezete az 5.12. ábrán látható.

```
private static IEnumerable<double> GetBand(Datapoint datapoint, IEnumerable<double> splits)
{
    List<double> result = new List<double>();
    var enumerator = splits.GetEnumerator();
    enumerator.MoveNext(); // otherwise first value is 0
    var prev = enumerator.Current;
    result.Add( GetDecibelSum(datapoint.data.Take((int)(prev / datapoint.freq))));
    while (enumerator.MoveNext())
    {

        result.Add(GetDecibelSum(datapoint.data
                                 .Take((int)(enumerator.Current / datapoint.freq))
                                 .Skip((int)(prev / datapoint.freq))
                                 ));
        prev = enumerator.Current;
    }
    result.Add(GetDecibelSum(datapoint.data.Skip((int)(enumerator.Current / datapoint.freq))));
    return result;
}
```

5.12 ábra DataConverter.GetBand függvény

A függvény csak a kiszámított értékeket adja vissza visszatérési értékként. A sávok középértékének lekéréséhez egy külön függvényt tartalmaz, ami csak egy logikai változó kér bemeneti paraméterként, igaz érték esetén a harmadoktáv középértékeket adja vissza, hamis érték esetén pedig az oktávoknak a középértékét.

6. Tesztelés

A tesztelés szerves részét képezi a fejlesztési folyamatnak. Ebben a fejezetben az általam használt tesztelési eszközöket, technológiákat fogom leírni.

6.1 Kliens

A kliensprogram fejlesztése során többször változott a tesztelési metodika attól függően, hogy éppen mire volt szükség. A fejlesztés kezdeti fázisában, amikor Python segítségével próbáltam megoldani a problémát, Pyplot segítségével rajzoltattam ki a feldolgozott adatot futásidőben. Miután elkezdtem .NET-ben fejleszteni a programot, a Visual Studio-nak az "ArrayPlotter64" kiegészítőjét használtam hasonló célokra. Ez a kiegészítő lehetővé tette, hogy fejlesztés közben egy tömb értékeit megjelenítsem vizuálisan, egy grafikonon. A kód módosítását nem igényelte ehhez a kiegészítő, az értékeket a memóriából olvasta ki futásidőben. Annyira volt szükség ehhez, hogy Hibakeresési üzemmódban legyen futtatva a program a Visual Studio keretein belül.

6.2 API

Az API fejlesztése közben fontos volt, hogy a lehető leghatékonyabban lehessen ellenőrizni a működését. Ehhez Postman-t használtam, ami egy direkt API tesztelésre kifejlesztett alkalmazás. Az alkalmazásban létre lehet hozni különböző gyűjteményeket, és ezeken a gyűjteményeken belül különböző API kéréseket tudunk létrehozni. Ezeknél az API kéréseknél be tudunk állítani minden paramétert, amit egy API lekérésnél be lehet állítani. A fejléceknél automatikus létrehoz egy pár olyan értéket, ami könnyebé teheti a tesztelést, így nem kell manuálisan beírjunk olyan fejléceket, amiket a böngésző tölt ki egy normális esetben. Sokat használtam fejlesztés közben, amikor valamit változtattam az API szerver kódjában és meg kellet nézzem, hogy az érintett kéréseknél megfelelően viselkedik-e a szerver. Akkor is sokat használtam, amikor a weboldal felületét fejlesztettem és valami nem működött megfelelően, így könnyedén ki tudtam deríteni, hogy a lekérés paramétereiben van valamilyen probléma, és a Postman klienssel se kapom meg a kívánt választ, vagy működik vele, és a weboldalon belül megy félre valami. Használtam az alkalmazás automatizált tesztjeit is. Az alkalmazásban le lehet programozni teszteket, amiben definiáljuk, hogy mit várunk az API szervertől. Akár azt, hogy érkezzen sikeres válasz a kérésre, hogyan nézzen ki a válasz, vagy hogy mi az az időlimit, amin belül szeretnénk választ kapni. Ezeket az automatizált teszteket azonban a fejlesztés során fedeztem fel, és miután elkezdtem használni, már nem tudtam sokszor igénybe venni.

6.3 Weboldal

A mai világban elengedhetetlen, hogy egy weboldal az asztali gépek széles vásznán, és a mobil eszközök keskeny kijelzőjén is egyaránt jól működjön. A weboldal tesztelési folyamata jelentősen felgyorsult, köszönhetően az olyan modern fejlesztési eszközöknek, mint a "Vite", ami egy két részből álló szoftveres eszköz. Az első egy fejlesztéshez használt webszerver. A Vite szerver használatával a kód módosítását követően azonnal lehet látni a frissítéseket, nem kell kézzel frissíteni az oldalt. A második funkciója a szerver élesítésénél jön szóba, amikor végeztünk a fejlesztési folyamat egy ciklusával, és szeretnénk kitenni, az erre használt szerverre, a weboldalunkat, akkor a Vite segítségével könnyedén le tudjuk generálni a statikus fájlokat, amiket már csak a megfelelő helyre kell másolni a szerveren.

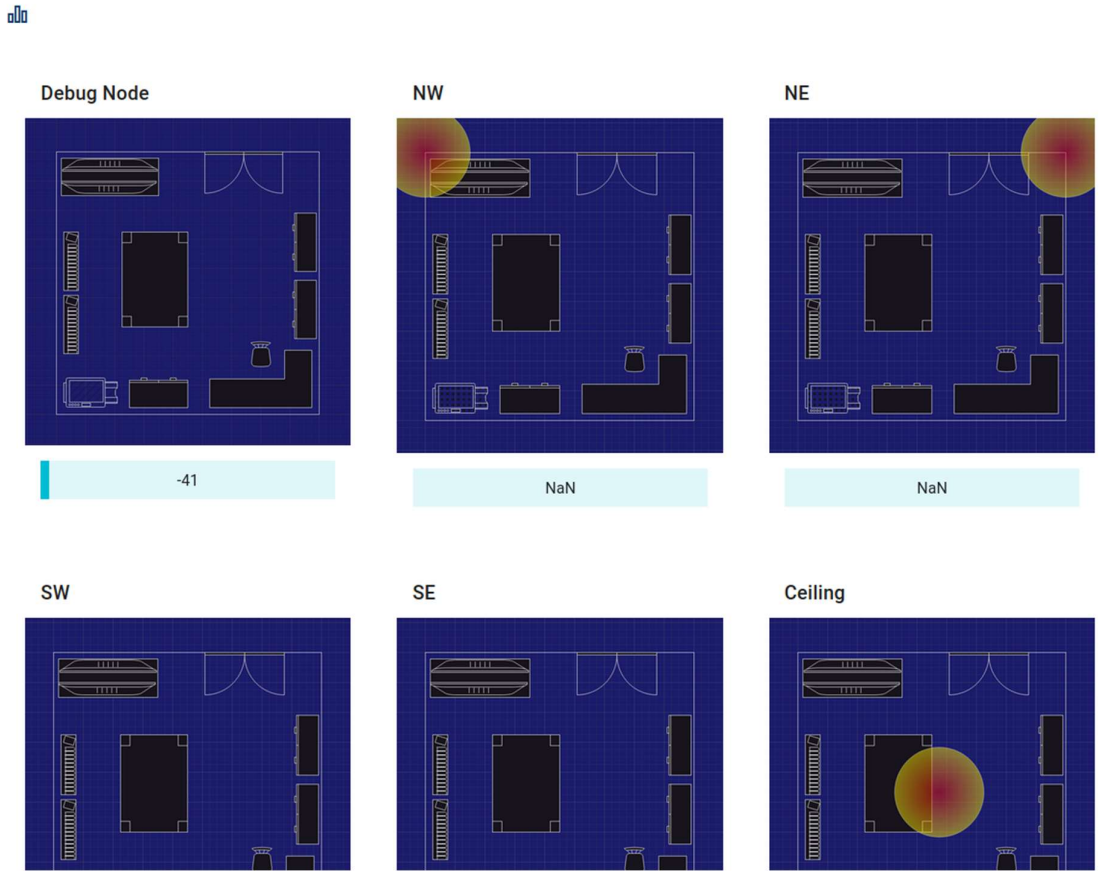
A weboldal fejlesztése során részletesen kielemeztem, hogy a kinézet hogyan tölti ki a rendelkezésre álló teret, különösen a törési pontok környékén, és a legelterjedtebb mobil eszközök felbontásában. A Chrome fejlesztési eszköztár ebben nagy segítség volt, lehetővé téve, hogy egy kattintásra kiválasszam ezeknek az eszközöknek a felbontását, és lehetővé tette a mobil platform imitálását is.

A fejlesztés során az API szerver szimulálásához json-servert használtam, ez a szoftver lehetővé tette, hogy a weboldalt tudjam teljes körűen tesztelni azután is, hogy elkezdtem API lekéréseket használni benne, anélkül hogy a szerverre feltelepítettem volna a tesztelés alatt álló weboldalt. A szoftver beolvas egy JSON fájlt, amibe előzőleg beleírtuk az adatbázis szerkezetét, és ehhez megfelelő lekérdezéseket generál és kiszolgál. Egy kapcsoló segítségével beállítható a port amit használ, így az API szerverre mutató linkek kicserélése nélkül tudtam tesztelni a weboldalt.

7. Felhasználói útmutató

7.1 A weboldal használata

A weboldal megnyitásakor az összes használatban lévő eszköz látható, listába rendezve. Erről egy mintát a 7.1. ábrán látható.

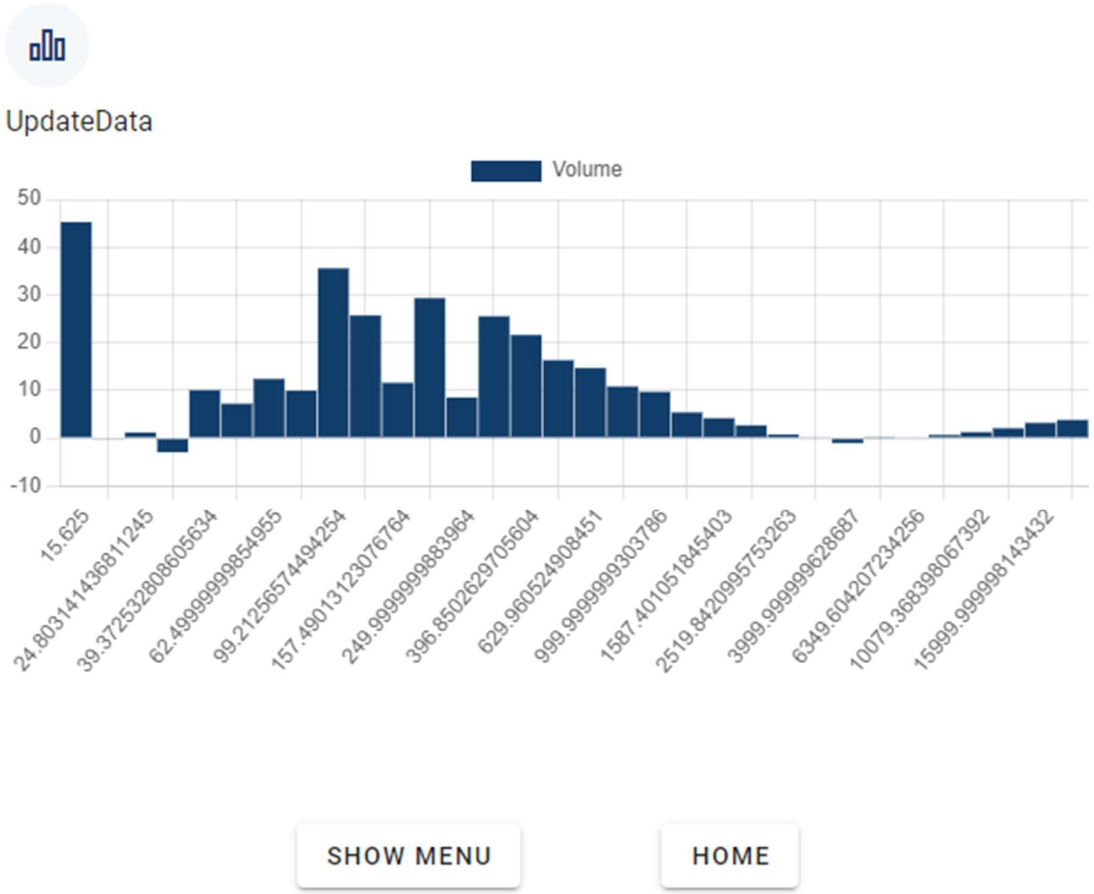


7.1. ábra főoldal kinézete

Egy ilyen eszközre ránézve látható az eszköz neve, egy kép, ami segíthet beazonosítani azt, és alatta egy csík, ami az utoljára érzékelt zajszintet jelöli.

Az egyik eszközre kattintva, az oldal átvisz minket a részletes nézethez. Ehhez a eszközhöz tartozó adatok bármelyik részére lehet kattintani.

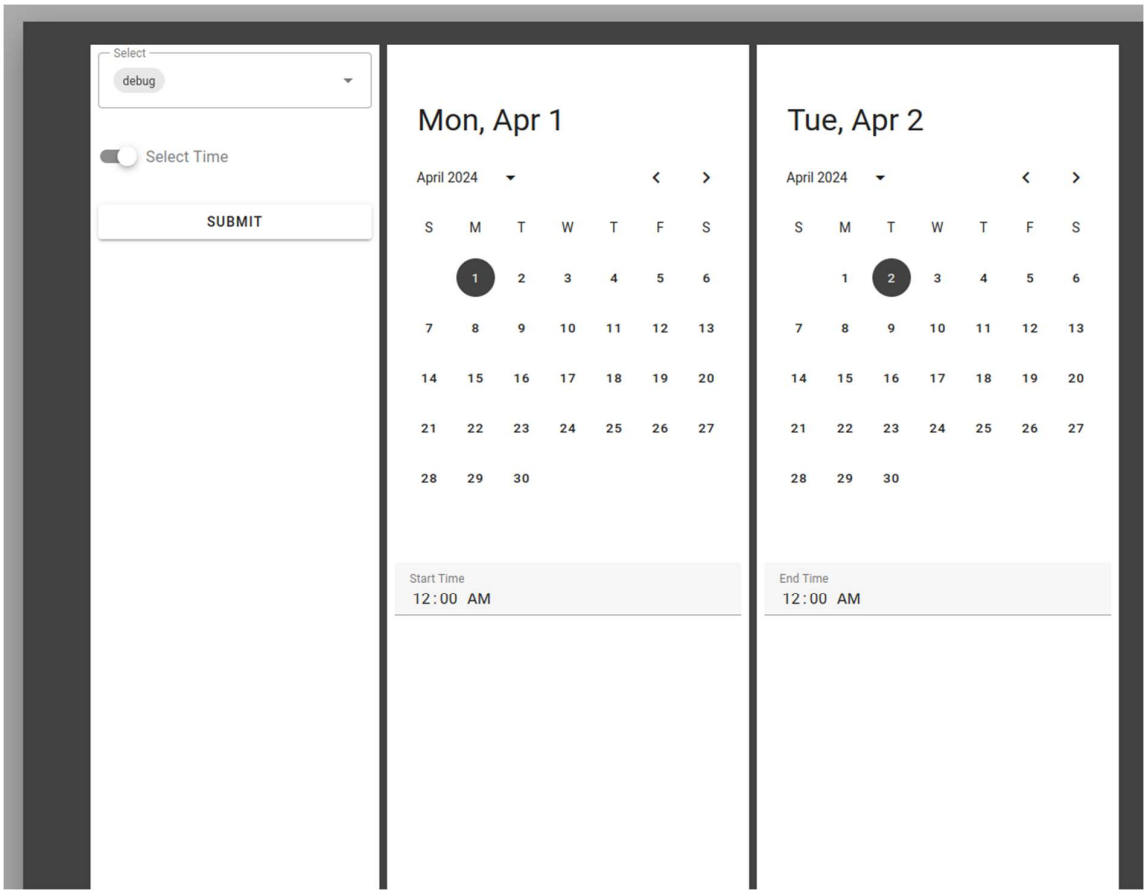
Ezen az oldalon láthatunk egy grafikont, ami a mutatja a mért zajszintet decibelben, frekvenciákra bontva. A grafikon periodikusan frissíti magát. Az adatok manuális frissítéséhez használhatjuk a grafikon bal felső sarkában található gombot is.



7.2. ábra részletes nézet

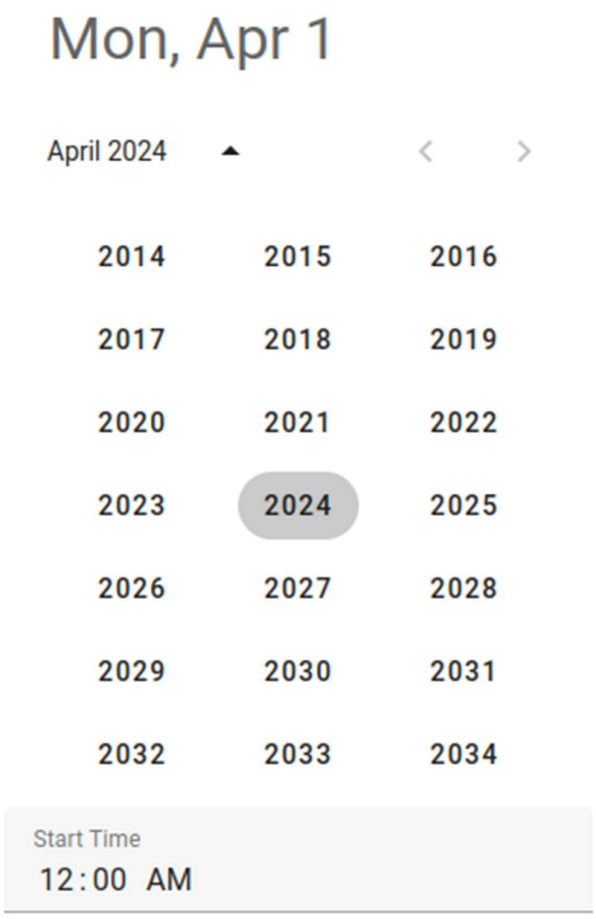
A pontos mért zajszint megtekintéséhez elegendő az oszlop fölé vinni a kurzort, és a weboldal kiírja az ahhoz a frekvenciához tartozó pontos értéket.

A zajszint egy korábbi állapotának a megtekintéséhez, a "show menu" gombra kattintva megnyílik egy menü, ahogy az a 7.3. ábrán is látható.



7.3. ábra menü nézet

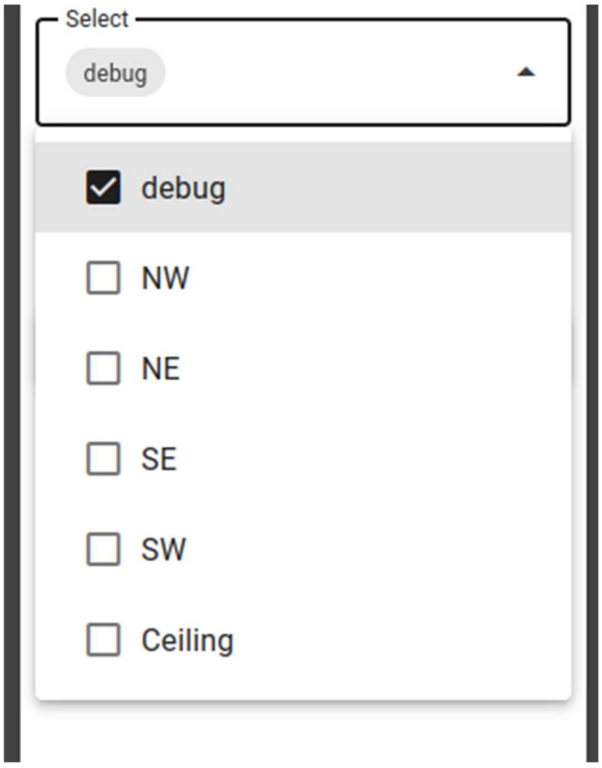
Ebben a menüben, a baloldali dátum választó felület segítségével, kiválasztható a lekérdezni kívánt időszakasz kezdete. Az időpontot a dátum választó alatt lehet megadni. A jobb oldali felület felépítése szerkezetileg azonos, azonban ezzel a lekérdezni kívánt időszakasz vége választható ki. Az évszám megváltoztatásához található egy nyíl az évszám mellett, a bal felső sarokban. Erre rákattintva előjön egy lenyíló menü. Ennek a menünek a kinézete a 7.4. ábrán megtekinthető.



7.4. ábra Év választó menü

A hónapok között váltáshoz, a jobb felső sarokban található nyilakat lehet használni.

A menü bal oldalán kiválasztható az eszköz azonosítója egy lenyíló lista formájában, ami a 7.5. ábrán látható. Több eszköz kiválasztása esetén az összes kiválasztott eszköz átlag mérését fogjuk látni.



7.5. ábra eszköz választó menü

Az alatta található kapcsoló segítségével eltüntethető az időpont választó lehetőség a dátum választó alatt. Ilyenkor egész napokat tudunk lekérdezni.

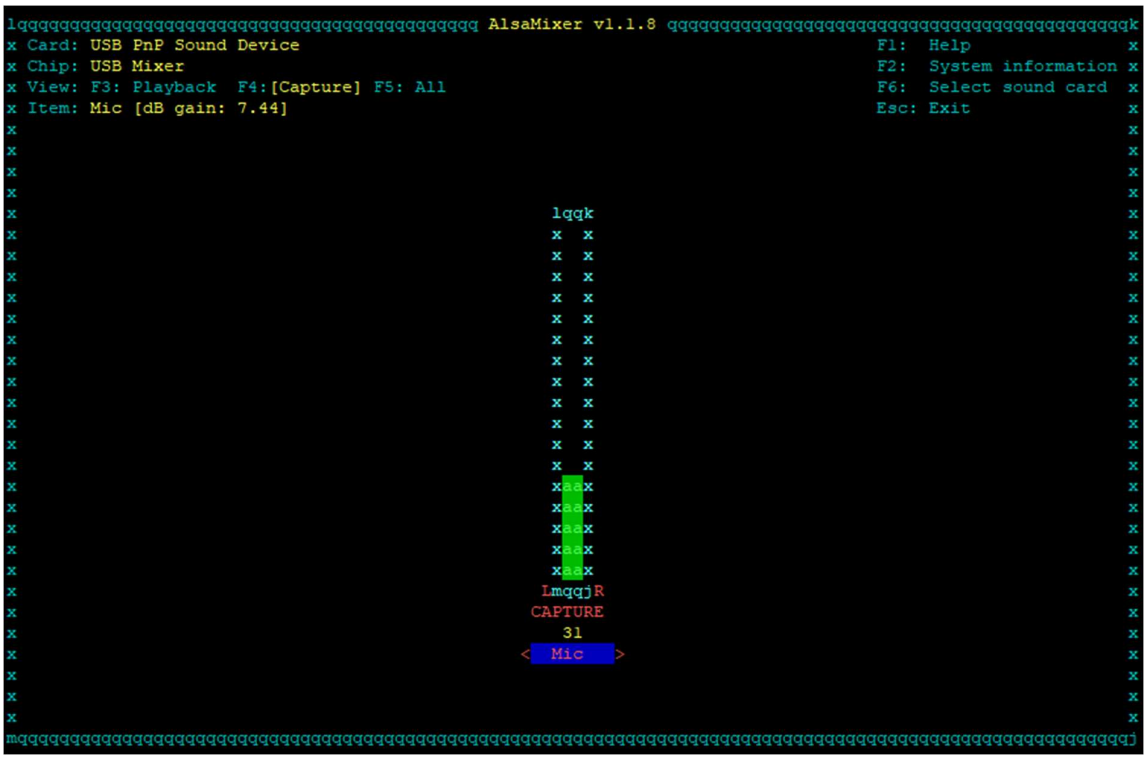
A "submit" gombra kattintva a weboldal elkezdi az adat lekérését, és bezárja a menüt. A menüből való kilépéshez asztali gépen az escape gomb használható, illetve a menü mellé kattintva az oldal szintén bezárja a menüt. Mobil eszközön ugyanehhez a vissza gombot használhatjuk.

A weboldal böngészése során, a bal felső sarokban található ikonra kattintva bármikor a főoldalra visz minket vissza az oldal.

7.2 Kliens program telepítése és használata

A kliens program használatához szükség van egy Raspberry Pi 4 Model B eszközre. Más ARM architektúrájú számítógépen is működhet a szoftver, azonban a program futása csak az előbbi típusú eszközön lett tesztelve. Eltérő típusú eszközön a felhasználónak kell meggyőződni arról, hogy a program megfelelően tud működni az adott környezetben, és

a továbbiakban leírtak a korábban említett eszközre vonatkoznak. Szükség van továbbá egy microSD memória kártyára, amire már fel van telepítve egy Linux disztribúció, ehhez ajánlott a legújabb Raspberry Pi OS, ami egy Debian alapú rendszer. A program működéséhez fontos, hogy legyen egy USB hangkártya és egy mikrofon, amit az eszközhöz lehet csatlakoztatni. Miután az eszköz be van üzemelve, a következő lépés ellenőrizni, hogy a rendszer felismerte-e a mikrofont. Ezt az alsamixer parancs segítségével lehet megtenni. A 7.6. ábrán látható egy példa, ahol sikeresen felismerte a mikrofont az eszköz



7.6. ábra alsamixer

Itt az F4 billentyűt lenyomva legalább 1 Felvevő eszközt kell látni. Miután az eszköz hálózati beállításainak a konfigurálása megtörtént, a Programot kell feljuttatni az eszközre. Ehhez érdemes valamilyen hálózati átviteli protokollt alkalmazni, például SCP-t. Az eszközre másolt fájlt ezután a "chmod +x" paranccsal kell futtathatóvá tenni. Ezután a megfelelő paraméterekkel el lehet indítani a programot. A program paraméterezéséhez

egy config.yaml fájlt kell létrehozni a program mappájában. A létrehozott fájlban a következő paramétereket tudjuk beállítani:

- **devicenumber**: ez a paraméter határozza meg, hogy melyik eszközt használja a program, és ez várhatóan egybevág az alsamixer harmadik fülén található eszközök közötti sorszámával 0-tól kezdve.
- **samplingrate**: a használt mintavételezési frekvencia, érdemes a hangkártya natív értékére beállítani.
- **clientid**: az API szerver elérési útvonala.
- **buffersize**: az adatfeldolgozási szakaszok hossza, a minták számában megadva.
- **server**: A szervernek az elérési útvonala. tartalmazza a protokollt, a tartománynevet, a portot és az útvonalat is, például "http://example.org:5000/api/audio"

A program automatikus indításához egy ütemezőt, vagy szolgáltatáskezelőt lehet használni, például crontab vagy systemd. Ahhoz, hogy systemd segítségével hogyan lehet létrehozni egy szolgáltatást, a Szerver telepítése részen belül található egy példa.

7.3 Szerver telepítése

Az alkalmazás teljes körű beüzemeléséhez szükséges a szerveroldali szoftver telepítése. Ehhez szükség van egy szerverre, ez lehet fizikai vagy virtuális szerver is. A szerver telepítésének megkezdése előfeltétele, hogy már fel van telepítve egy Linux disztribúció és a hálózati beállítások be vannak konfigurálva. A szerver telepítéséhez szükséges ezenkívül egy SQL adatbázis és az ehhez tartozó "ConnectionString", ami az adatbáziselérést biztosítja. Szükséges egy webszerver feltelepítése is, a statikus tartalmak kiszolgálásához. Erre a célra ajánlott az NGINX használata, ez mivel használható fordított proxyként is, gördülékennyé téve az API szerver és a webszerver ugyanazon szerveren való használatát.

Az API szerver és a weboldal fájljainak a szerverre juttatásához célszerű egy hálózati fájlátviteli protokollt használni, például SCP-t. Miután a fájlok a szerverre kerültek, a weboldalhoz tartozó statikus fájlokat a /var/www/html mappába kell tenni, az API szerver fájljait pedig egy tetszőleges mappába.

7.3.1 API

Az API szerverhez használt adatbázis elérést az appsettings.json fájl módosításával lehet beállítani. A kliens programhoz hasonlóan, miután felkerültek a fájlok a szerverre, a "chmod +x" parancs segítségével lehet futtathatóvá tenni az alkalmazást, ami ebben az esetben a az API001 fájl.

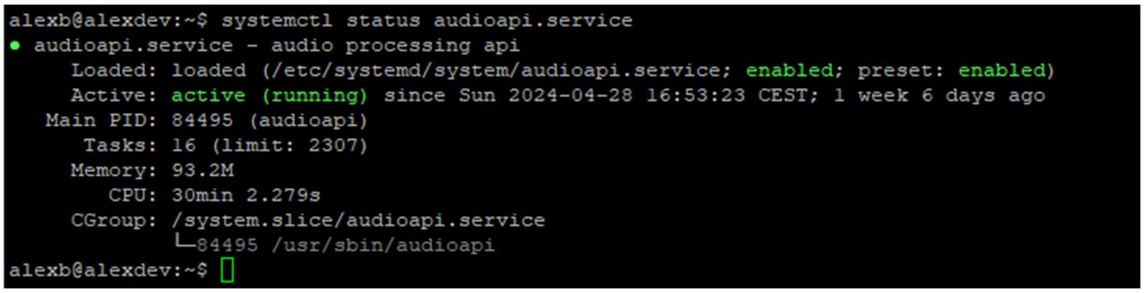
7.3.2 Systemd szolgáltatás

Az API szerver üzemeltetéséhez érdemes létrehozni egy szolgáltatást, így azt a webszerverhez hasonló módon lehet kezelni a későbbiekben. Az első lépés ehhez egy szimbolikus fájl létrehozása a /usr/sbin/ mappában, ami az alkalmazásra mutat. Az ln parancs segítségével hozható létre egy ilyen szimbolikus fájl, például ha az API szerver fájljai a /home/alexb/api/ mappába kerültek, akkor az "ln -s /home/alexb/api/ /usr/sbin/audioapi" parancs segítségével lehet létrehozni egy audioapi nevű szimbolikus linket, ami a tényleges alkalmazásra mutat. A szimbolikus fájl létrehozását követően, a "/etc/systemd/system/" mappában kell létrehozni egy "audioapi.service" fájlt. A létrehozott fájlban tudjuk bekonfigurálni a szolgáltatást, a lenti ábrán látható módon.

```
[Unit]
Description=audio processing api
[Service]
WorkingDirectory=/home/alexb/api
ExecStart=/usr/sbin/audioapi
Restart=always
RestartSec=10
KillSignal=SIGINT
Environment=ASPNETCORE_ENVIRONMENT=Production
[Install]
WantedBy=multi-user.target
```

7.7. ábra audioapi.service

Ezután létrejön az audioapi.service szolgáltatás, amit a későbbiekben a systemctl segítségével lehet kezelni.



7.8. ábra helyesen működő szolgálatás

7.3.3 Nginx

Ahhoz, hogy az Nginx fordított proxyként működjön, és a statikus fájlokat elérhetővé tegye, az API kéréseket pedig továbbítsa az API szerver felé, be kell konfigurálni azt, a /etc/nginx/nginx.conf fájl segítségével. Ebben a fájlban be kell állítani a két útvonalat, amiből az egyik a weboldal fájljaira mutat, a másik pedig az API elérési útvonalára. Az API esetében meg kell adni egy pár fejlécet, hogy megfelelően működjön. Az alábbi ábrán láthatóak ezek.

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
error_log /var/log/nginx/error.log;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 768;
        # multi_accept on;
}

http {
  map $http_connection $connection_upgrade {
    "~*Upgrade" $http_connection;
    default keep-alive;
  }

  server {
    listen        80;
    server_name _;
    location / {
        root /var/www/html;
        include  /etc/nginx/mime.types;
        try_files $uri $uri/ /index.html;
    }
    location /api {
        proxy_pass         http://localhost:5000/api;
        proxy_http_version 1.1;
        proxy_set_header   Upgrade $http_upgrade;
        proxy_set_header   Connection $connection_upgrade;
        proxy_set_header   Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header   X-Forwarded-Proto $scheme;

    }
  }
}
```

7.9. ábra alsamixer

8. Összegzés

A szakdolgozat elkészítése során sokat tanultam a modern keretrendszerekről, és a webes fejlesztéshez használható technológiákról, ami eddig egy viszonylag ismeretlen terület volt számomra. A modern webfejlesztés szempontjai alapján próbáltam egy könnyen

használható felületet fejleszteni a szakdolgozatomhoz. Megismerkedtem a API fejlesztés során előforduló akadályokkal, és a fejlesztést segítő eszközökkel.

A szoftver a kitűzött célokat (véleményem szerint) kielégíti, miszerint a rendszer képes kell legyen zajszint mérése, és ezeknek az adatoknak a megfelelő megjelenítésére. A fejlesztés során sok fejlesztési ötlet jutott eszembe, ami alapján akár több irányba is el tudom képzelni a szoftvernek a továbbfejlesztését.

Különböző frekvencia súlyozások implementálása lehetővé tenné a felhasználó számára, hogy jobban igazodjanak azokhoz a szabályozásokhoz, amik vonatkoznak rá.

A zajszint időbeni kimutatása egy hasznos kiegészítés lehetne, amennyiben erre igény van. Ez egy másik grafikon formájában működhetne, ami az összesített zajszintet mutatná egy meghatározott ideig visszamenőleg. Ez megkönnyítené a különböző ciklikus trendek analizálását.

Aggregációs táblák bevezetése felgyorsíthatná a lekérdezéseket. Ez azt jelentené, hogy lenne egy-egy olyan tábla, ami órás, napos felbontásban tárolná az adatokat. Ezek a táblák tartalmaznának egy súlyozást, azaz az adott időszakban történt mérések számát. Amennyiben ezzel az eredeti adat által elfoglalt helyet is szeretnénk visszanyerni, különböző aggregációs függvények eredményét is szükséges lenne eltárolni, mint például a maximum értéket.

Az eddigi fejlesztési ötletek talán jobban befolyásolnák a program központi működését, a következőkhöz képest, de a következők tovább javítanák a rendszer használhatóságát:

- értesítési rendszer implementálása
- központi működés lehetősége (ezt mindenképp az "on-premise" üzemeltetés mellet, egy plusz opcióként tudom elképzelni)
- kliensek központi kezelése, frissítése
- felügyeleti rendszerek számára SNMP támogatás biztosítása
- több kliens eszköz integrációja, az API segítségével

9. Summary

While writing this thesis, I learned a lot about modern frameworks and technologies for web development, which was a relatively unknown field for me. I tried to develop an easy to use interface for my thesis based on the aspects of modern web development. I learned about the obstacles that can occur during API development and the tools that can help me develop them.

The software satisfies (in my opinion) the stated goals, which is that the system should be able to measure noise levels and display these data appropriately. During the development process, I came up with many ideas for improvement, which I can imagine further development of the software in several directions.

Implementing different frequency weightings would allow the user to better adapt to the regulations that apply to him.

A temporal detection of the noise level could be a useful addition, if there is a need for it. This could take the form of another graph showing the aggregate noise level over a specified time period. This would facilitate the analysis of different cyclical trends.

The introduction of aggregation tables could speed up queries. This would mean that there would be tables that would store data at hourly, daily resolution. These tables would contain a weighting, i.e. the number of measurements taken in a given period. If this is implemented to recover the space occupied by the original data also, it would also be necessary to store the results of various aggregation functions, such as the maximum value.

The ideas for improvement so far might have more impact on the core operation of the software than the following, but the following would further improve the usability of the system:

- implementation of a notification system
- possibility of centralised operation (I can only see this working as an additional option to on-premise operation)
- central management and updating of clients
- provide SNMP support for monitoring systems

- integration of different client devices, via API

10. IRODALOMJEGYZÉK

[1] H. B. Csaba, MySQL.NET - MySQL Server adatbázis-programozás .NET környezetben, BBS-INFO, 2007.

[2] Google, "Material Design," [Online]. Available: https://m3.material.io. [Accessed 14 05 2024].

[3] P. Marino, Mastering c♯ and .net programming, Packt Publishing Limited, 2016.

[4] P. community, "PortAudio: Main Page," [Online]. Available: https://www.portaudio.com/docs.html. [Accessed 14 05 2024].

[5] S. W. Harden, "Plot Audio FFT with C#," [Online]. Available: https://swharden.com/csdv/audio/fft/. [Accessed 12 05 2024].

[6] L. Julia, Programming Entity Framework: Dbcontext: Querying, Changing, and Validating Your Data with Entity Framework, Oreilly Media, 2012.

[7] J. Mayo, LINQ programming, McGraw-Hill, 2009.

[8] Mozilla, "Cross-Origin Resource Sharing (CORS)," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS#what_requests_use_cors. [Accessed 12 05 2024].

[9] SVANTEK, "Sound Pressure Level (SPL) | Svantek Academy," [Online]. Available: https://svantek.com/academy/sound-pressure-level-spl/. [Accessed 14 05 2024].

[10] A.-Y. John, Vue.js 3 By Example: Blueprints to learn Vue web development, full-stack development, and cross-platform development quickly, Packt Pub, 2021.