**Eötvös Loránd Tudományegyetem**

**Informatikai Kar**

**Informatikatudományi Intézet**

**Programozáselmélet és Szoftvertechnológia Tanszék**

Wireless Tools Analytics

Szerző: Témavezető:

Bezzeg András Benjamin Dr. Nikovits Tibor

Programtervező informatikus BSc. mesteroktató

**Budapest , 2023**

Ide kerül a hivatalos témabejelentő lap.

Tartalomjegyzék

[1. Bevezetés 1](#_Toc154416551)

[1.1. Rövid kontextus: 1](#_Toc154416552)

[1.2. Motiváció: 1](#_Toc154416553)

[1.3. Az alkalmazás: 2](#_Toc154416554)

[2. Felhasználói dokumentáció 3](#_Toc154416555)

[2.1. Projekt struktúra 3](#_Toc154416556)

[2.1.1. /db/ 3](#_Toc154416557)

[2.1.2. /server/ 4](#_Toc154416558)

[2.1.3. /client/ 5](#_Toc154416559)

[2.2. Futtatási lépések: 6](#_Toc154416560)

[2.2.1. Leírás: 7](#_Toc154416561)

[2.2.2. run-script.js 8](#_Toc154416562)

[2.2.3. Extra tudni valók: 8](#_Toc154416563)

[2.3. Felhasználói felület: 9](#_Toc154416564)

[2.3.1. Beállítások 9](#_Toc154416565)

[2.3.2. Navigálás 9](#_Toc154416566)

[2.3.3. Total page 9](#_Toc154416567)

[2.3.4. Per Tool page 11](#_Toc154416568)

[2.3.5. Compare page 11](#_Toc154416569)

[3. Fejlesztői dokumentáció 13](#_Toc154416570)

[3.1. Leírás 13](#_Toc154416571)

[3.1.1. Verziókezelés 13](#_Toc154416572)

[3.2. Projektstruktúra leírása: 13](#_Toc154416573)

[3.2.1. /db/ 14](#_Toc154416574)

[3.2.2. /server/ 15](#_Toc154416575)

[3.2.3. /client/ 15](#_Toc154416576)

[3.3. Futtatási lépések fejlesztőknek: 15](#_Toc154416577)

[3.4. Fejlesztési terv- és folyamatok: 15](#_Toc154416578)

[3.4.1. Technológiák 15](#_Toc154416579)

[3.5. Adatbázisrétegek 16](#_Toc154416580)

[3.5.1. Docker file 16](#_Toc154416581)

[3.6. Szerver 16](#_Toc154416582)

[3.7. Kliens 17](#_Toc154416583)

[3.8. Adatbiztonság és Teljesítmény 17](#_Toc154416584)

[3.9. Összegzés 17](#_Toc154416585)

[4. Összefoglalás és további fejlesztési lehetőségek 18](#_Toc154416586)

[5. Irodalomjegyzék 19](#_Toc154416587)

[6. Melléklet 20](#_Toc154416588)

# Bevezetés

## Rövid kontextus:

A cégünk vezetéknélküli kommunikációs chipek tervezésével , szabadalmaztatásával és gyártásával foglalkozik. Mindemellett biztosit egy integrált fejlesztési környezetet ahol a felhasználó tesztelheti ,megfigyelheti, konfigurálhatja vagy egyéni kódot is írhat ezekre a hardware-ekre. (Továbbiakban ***Studio*** néven erre referálok).

Ebben a Studio-ban számos saját és globális vezetéknélküli kommunikációs protokol támogatására van lehetőség, azonban nem mindenki képes hosszú és bonyolult C kód írására, a cég ezeket a felhasználókat is figyelembe véve hozta létre a *Wireless* *Tooling* csapatot aminek énis a tagja vagyok.

A célunk átlagos vagy kevésbé hozzáértő felhasználók részére is lehetővé tenni, hogy egyszerűen generálhassanak hardware kódot. Ehhez különböző webes applikációkat hozunk létre ami az adott technológiához tartozó kódot képesek generálni egy SDK segítségével. Ezeket a webes applikációkat hívjuk ***Tool-***oknak.

## Motiváció:

Ezek a **Tool**-ok tehát egy külső integrált fejlesztési környezetbe épített webes alkalmazások, aminek a használatáról naplózás készül egy felhőben lévő adatbázisba. Ezeket a naplózásokat fogom **Esemény**-nek hívni a továbbiakban

A képen szöveg, képernyőkép, menü, monokróm látható

Automatikusan generált leírás

1. ábra: "Esemény" minták

Mivel ezeknek a Tool-oknak a száma, mára bőven kétszámjegyűvé nőtt, igény alakult ki, ezen naplózások feldolgozására. A cél, hogy feltérképezhessük és láthatóvá tegyük a felhasználói szokásokat és hogy melyik projekt milyen súlyú hasznot hoz a belefektetett munka arányában.

Ennek a kivitelezésével lettem megbízva a gyakornoki időszakom alatt és ez adta a szakdolgozat témáját.

## Az alkalmazás:

A célom tehát egy olyan alkalmazás elkészítése volt, ami ezeket az adatokat felhasználva segíti és információval látja el először a csapatatot majd a menedzsmentet. Ehhez szintén egy webes alkalmazást fogok elkészíteni, ami egy belső hálózaton való elérésre lesz alkalmas.

Tehát felhasználók kizárólag cég alkalmazottak akik nagy része szakmabeli felhasználó.

Kisebb-nagyobb eltérések akadnak az eredeti verzióval szemben. Az adatokat véletlenszerűen generáltattam apró átalakításokkal és jelentősen kisebb mennyiségben.

A képen szöveg, képernyőkép, Diagram, szoftver látható

Automatikusan generált leírás

2. ábra: Alkalmazás minta

A projekt struktúrájának kialakításakor és a technológiák kiválasztásakor a standalone lokálisan futtathatóság és a teljes és egyszerű verziókezelés volt a fő szempont.

# Felhasználói dokumentáció

## Projekt struktúra

A képen szöveg, képernyőkép, Betűtípus, szoftver látható

Automatikusan generált leírás

3. ábra: Forráskód mappa struktúra

### /db/

Ebben a mappában találhatóak az adatbázis elindításához szükséges file-ok. A legfontosabb egyből a gyökérben lévő **docker-compose.yml**, ami a docker konténer leíró file-ja és a docker-comopose CLI parancs alapértelmezett inputja. Ennek tartalmát a fejlesztői dokumentációban részletezem. Ebben található egy mock nevezetű mappa is , itt találhatóak az inicializáló SQL szkriptek. Két almappával rendelekezik:

* *db/mock/data:* Ennek a mappának a tartalma automatikusan felkerül az elindított konténerbe és a benne lévő összes \*.sql kiterjesztésű file futtatásra kerül „abc” sorrendben. Ezek alkotják az „alap” tábláinkat.
* *db/mock/runtime-scripts:* Ennek a mappának a tartalma szintén felkerül egy előre definiált mappába a konténeren belül , azonban a futtatására manuálisan van szükség. Ezek alkotják a „kisegítő” tábláinkat.

A képen szöveg, képernyőkép, Betűtípus, tervezés látható

Automatikusan generált leírás

4. ábra: Db mappa lenyitva

### /server/

Itt található a szerver futtatásához tartozó forráskód. Ezt is a fejlesztői dokumentációban fogom kellően részletezni. A legfontosabb file a gyökérben található „main.ts”, amit előszőr lekell fordítani tiszta javascript kódra , majd futtatásával indíthatjuk el a szerver szolgálltatást alapértelmezetten a 9000-es porton.

A képen szöveg, képernyőkép, szoftver, Betűtípus látható

Automatikusan generált leírás

5. ábra: Lenyitott server mappa

### /client/

Itt a tool\_analytics\_frontend mappában található egy quasar struktúrájó frontend projekt ami a kliensként szolgál. Részletezni itt is a fejlesztői dokumentációban fogok, de minden a framework struktúráját követi aminek a részletes dokumentációja elérhető a <https://quasar.dev/> című honlapon.

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás

6. ábra: Lenyitott client mappa

## Manuális futtatás:

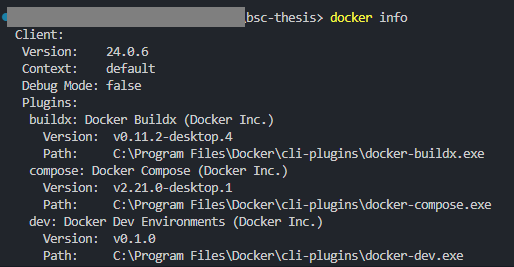
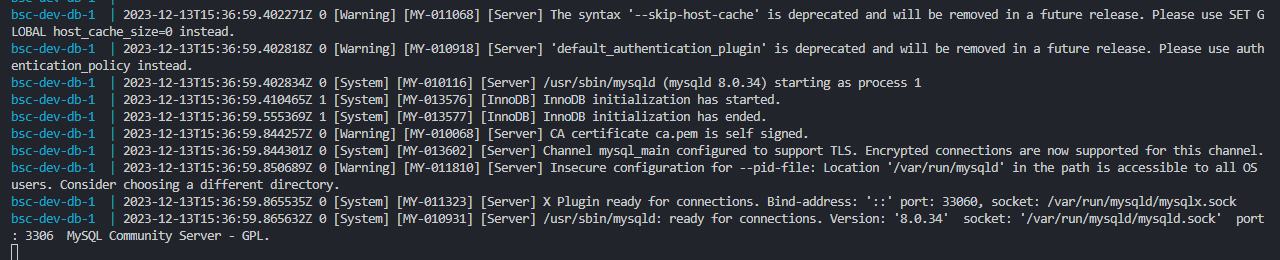
A kicsomagolást követően a wireless\_tools\_analytics mappában találhatóak az alkalmazás forráskódjai. Három fontos dologgal kell rendelkeznie a felhasználónak:

* Node.js futtatási környezet
  + <https://nodejs.org/en/download>
* Npm package manager
  + <https://docs.npmjs.com/downloading-and-installing-node-js-and-npm>
* Docker CLI ( windows és mac esetében )
* Futó docker daemon
  + Docker-Desktop
    - <https://www.docker.com/products/docker-desktop/>
  + Rancher Desktop , ha gond a liszenszelés
    - <https://rancherdesktop.io/>

Minden más esetleges dependenciát a docker vagy node package manager fog kezelni.

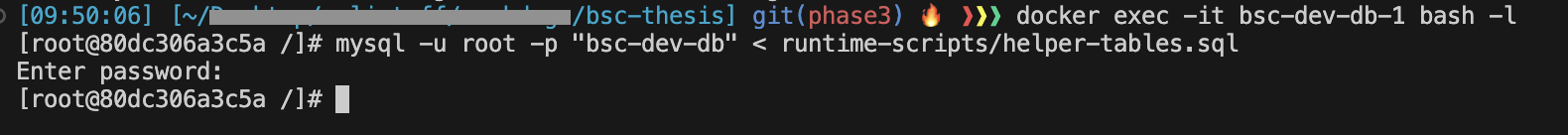
### Leírás:

A futtatás egy többlépéses parancssor eredménye lesz: TODO KÉPEK

1. Indítsuk el a docker-container-t
   1. Nyissunk meg egy terminált
   2. Győződjünk meg, hogy fut a docker daemon
   3. Navigáljunk a db mappába
   4. Adjuk ki a ’docker-compose up’ parancsot

7. ábra: Sikeres docker-compose utolsó üzenetei

1. Generáljuk le a segéd táblákat
   1. Sikeres docker container indítás után nyissunk egy új terminált ( az előzőt hagyjuk megnyitva!)
   2. Adjuk ki a `docker exec -it bsc-dev-db-1 bash -l` parancsot
   3. Ezzel megnyitottunk a konténerünkben egy terminált, itt adjuk ki a `mysql -u root -p "bsc-dev-db" < runtime-scripts/helper-tables.sql` parancsot
   4. Adjuk meg a docker-compose.yml-ben definiált jelszót („pwd” alapesetben)
   5. Sikeres futás esetén zárjuk be ezt a terminált



8. ábra: Sikeres runtime-scripts futtatás

1. Fordítsuk le és futtassuk a szerver kódot
   1. Navigáljunk a server mappába
   2. Győződjünk meg róla, hogy rendelkezünk node és npm parancsokkal
   3. Adjuk ki az `npm ci` parancsot
   4. Adjuk ki az `npm run start` parancsot , ez elvégzi mindkét lépést
   5. Siker esetén hagyjuk nyitva ezt a terminált is
2. Futtasuk a kliens kódot
   1. Navigáljuk a client/tool\_analytics\_frontend mappába
   2. Adjuk ki az `npm ci` parancsot
   3. Adjuk ki az `npm run dev` parancsot
   4. Siker esetén ezt a terminált is hagyjuk nyitva

### run-script.js

A fejlesztés elején célul tűztem ki, hogy ezt a bonyolult folyamatot a lehető legegyszerűbb formára szeretném hozni.

Ezért alternatívaként, készítettem egy extra modult. A szkrpit maga egy javascript-es szálkezelésre épülő szinkronizált parancsvégrehajtó modul, amivel a fenti parancsokat hajthatjuk végre egy terminálban. Szükséges a fent említett három dependencia ebben az esetben is.

Használatához csak lépjünk be a fő mappába egy terminálból és adjuk ki a `node runtime-script.js` nevű parancsot. A program tájékoztatni fog éppen melyik komponensen dolgozik és hogy ez milyen eredménnyel zárult, ha valami rész sikertelenül zárul, az egész leáll. Felszabadítja maga után a portokat, egy terminált foglal el és egyedül a docker container-t nem törli bezárás/ újra indításkor, azt manuálisan lehet meg tenni ha szükséges.

### Extra tudni valók:

A db mappa tartalmaz egy markdown file-t ahol a docker-specifikus extra információk találhatóak.

## Felhasználói felület:

A felület elsődleges célja a feldolgozott adatok interaktív és könnyen értelmezhető bemutatása-összehasoníltása. Minden oldal erre törekszik ,de más megközelítésből. Lehetőség van a teljes adathalmazból készült kimutatások megtekintésére, vagy Tool specifikusan megtekinti és összehasonlítani az adatokat. A felületen emellett pár kisegítő lehetőség is adott, mint például a nyelv válltás és sötét módra válltás és tooltip dobozok.

### Beállítások

Az oldalak tetején találhat egy sáv a címmel. Ennek a fenti sávnak a jobb szélén található egy fogaskerék gomb mely kattintásra megjeleníti a beállítások ablakot, amiben válthatunk sötét/világos mód és angol/magyar nyelv között. A sötét mód lecseréli a háttér színeket egy sötét árnyalatúra és minden szöveget fehérre állít.

A képen szöveg, Betűtípus, sor, képernyőkép látható

Automatikusan generált leírás

9. ábra: Beállítások ablak

### Navigálás

Az oldal tetején található egy sáv amiben a projekt címe látható. Kattintásra ez a cím hozza elő a navigációs ablakokat. Három oldal közül választhatunk, mindegyik más adatok megjelenítésére szolgál.

A képen szöveg, képernyőkép, Betűtípus, embléma látható

Automatikusan generált leírás

10. ábra: Navigációs menü

### Total page

A teljes eszköztárról készült kimutatásokat találhatjuk itt és ez szolgál jelenleg landoló oldalként. A tetején emiatt egy tájékoztató szöveg fogad minket , melle pedig egy álltalános jellemzése a feldolgozott adatoknak, mint például milyen időtartományban vizsgáltunk , milyen Tool-okat ,stb.

A képen szöveg, képernyőkép, szoftver, Weblap látható

Automatikusan generált leírás

11. ábra: Total oldal

Alatta pedig csoportosítva találhatjuk a különböző alapon (idő , operációs rendszer , hely) készített kimutatási diagrammokat. Minden diagram jobb felső sarkában található egy segítő dobozt ami leírja a diagramm célját és hogy milyen SQL parancs eredményéből jött létre a segédtáblája.

Látható egy jelen évi statisztika sáv is. Kattintásra ez lenyílik és különböző ez évi adatokra vonatkozó kis kártyákon mutat meg adatokat. Látható például melyik volt idén eddig a legnépszerűbb Tool. Minden kártya alján egy trend mutató ikon is található. Ez arra vonatkozik, hogy az itt látható adat az előző évhez képest, hogyan változott.

A képen szöveg, képernyőkép, Betűtípus, tervezés látható

Automatikusan generált leírás

12. ábra: Total oldal jelen évi adatok

Ezután a idő alapon vett diagrammok találhatóak. Kezdésnek egy teljes éves bontás Tool-onként. Mellette pedig egy összesített kiadott parancs diagramm található. Ezek alatt pedig választhatunk, hogy egy havi vagy egy napi bontásban szeretnénk látni ezeket az eseményeket.

A képen szöveg, Diagram, diagram, Betűtípus látható

Automatikusan generált leírás

13. ábra: Idő alapi bontás

### Per Tool page

Ezen az oldalon kiválaszthatunk egy Tool-t és csak a rá specifikus adatokból készült diagrammokat láthatjuk. Az oldal tetején van egy lenyitható mező amiben az elérhető Tool-ok listájából egyet kiválasztva a rá vonatkozó pár specifikus kimutatást láthatjuk dinamikusan. A legalsó földrajzi adatokat bemutató diagramm itt is lenyitásra jelenik meg.

A képen képernyőkép, szöveg, szoftver, diagram látható

Automatikusan generált leírás

14. ábra: Per tool oldal

### Compare page

Több kiválasztott eszköz összehasonlítására is lehetőség van. Az oldal tetején található buborékokból tetszés szerinti mennyiséget kiválasztva megjelennek meg a Tool-hoz köthető főbb jellemzők. (Teljes esemény szám, első használat éve, változási trend). Mellette pedig az összefelhasználásból kapott diagramm látható.

A képen szöveg, szoftver, diagram, Számítógépes ikon látható

Automatikusan generált leírás

15. ábra: Compare oldal

# Fejlesztői dokumentáció

## Leírás

Az Wireless Tools Analytics egy komplex (***háromrétegű kliens-szerver*** architektúrájú) rendszer, amely adatbázis-kezelést használva lehetővé teszi az adatok hatékony és strukturált kezelését. Az alkalmazás széles körű lehetőségeket biztosít az adatok lekérdezésére, feldolgozására és megjelenítésére különböző grafikonokon keresztül.

A tervezés során számos fontos kérdés merült fel a megvalósítást illetően. Ezek külön fel vannak tüntetve a rétegek részletes jellemzésénél. Fontos szempontnak találtam a könnyű és lokális kezelhetőséget ezért az alkalmazás kiegészült egy futtatást segítő script-tel is. Ez segíti az egyébként aszinkronikus parancsok szinkronizált végrehajtását.

### Verziókezelés

Verziókezelés git alapon a github oldalamon történt. A fő branch-em a main, ide került minden változtatás visszavezetve nagyobb mérföldkövek előtt. Emelet 3 phase elnevezésű branchen történt a projekt felépítése. Ezek egymásra épültek és a tervezés során elválasztott 3 fázist reprezentálják.

Számos apróbb branch is létrejött, ezek olyan fejlesztési vonatlaknak adnak lehetőséget, amik nem kötődnek szorosan a fő funkcionalitáshoz. Emiatt ideális esetben konflikus nélkül visszavezethető nagyobb branch-ekre.

## Projektstruktúra leírása:

Hasonlóan a felhasználói dokumentációban itt is 3 fő részre fogom osztani a bemutatást, azonban jobban részletezem a fejlesztéshez fontos komponensek felépítését , feladatait és megvalósításait.

### /db/

#### docker-compose.yml

A docker konténer konfiguráláshoz a gyökérben található docker-compose.yml file ad lehetőséget:

A képen szöveg, képernyőkép, szoftver látható

Automatikusan generált leírás

16. ábra: docker-compose.yml tartalma

Itt definiálhatóak a docker-compose up parancs kiadásakor a konténer és az abban futó service-k tulajdonságai. A példánkban megadtunk egy „db” nevű szervízt , ami egy mysql képfájl tartalmát és a hozzá szükséges dependenciákat fogja feltelepíteni.

Az enviroment alatt megadhatunk mysql specifikus tulajdonságokat is mint például az alapértelmezett adatbázisunk neve, és a belépéshez szükséges hitelesítési adatok.

A runtime script-ek futtatásánál az itt definiált jelszót fogja tőlünk kérni a konzol manuális indításnál.

A ports mezőkben adhatjuk meg , hogy a melyik portokon tegye elérhetővé a kommunikációt a konténer és.a hoszt között.

Volumes mezőben pedig olyan lokális mappák vagy file-k elérhetőséget adhatjuk meg melyekre szükségünk lesz a konténerben is. A példánkban két típusú volume használatára van példa. Az első egy mysql service specifikus volume , az „entrypoint”. Itt megadhatunk egy mappányi „\*.sql” kiterjesztésű file-t , amiknek a futtatását garantálja a docker inicializáló script-je miután a service maga létrejött. Ezzel tudjuk könnyeden biztosítani az adatbázisunk inicializálását. A másik egy álltalános másolást biztosít, az elől definiált hoszt mappa tartalmát felmásolja a konténer-ben megadott helyre, ezzel tudunk olyan scripteket végrehajtani amik futási időben vállnak lényegessé.

#### init scripts

A mock/data mappában találhatjuk az első volumehoz szükséges file-okat. Ez a mappa tartalmazza az adatbázis inicializálásához szükséges adatokat és a séma leírását.

#### runtime scripts

A mock/runtime-scripts mappában pedig a második volume-hoz szükésges file-ok vannak , ezeket a séma kialakítás és az adatok feltöltése után a segédtáblák létrehozásához használjuk.

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás

17. ábra: Runtime scripts részlete

### /server/

Itt találhatóak az express szerver futtatásához szükséges typescript file-ok. Ezeket fordítjuk vissza javascript file-okra futtatás előtt.

#### main.ts:

A fő futási szálunk itt történik, itt definiáljuk a express szervert és inkludáljuk minden dependeciáját.

#### connection.ts:

Itt definiáljuk a mysql kapcsolatunkat az express erre szolgáló plugin-jával. Ezt a kapcsolatot egy statikus változóként fogja használni a programunk.

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás

18. ábra: Connection.ts

#### types.ts

A typescript lehetőségeit használva előre definiálunk pár típust mint pl a diagrammok álltal beolvasott adatszerkezetet amiké alakítani fogjuk az adatbázis lekérdezéseket.

A képen szöveg, képernyőkép, szoftver látható

Automatikusan generált leírás

19. ábra: types.ts részlete

#### endpoints.ts

Itt soroljuk fel az express álltal kiszolgált API endpointokat, minden kliens álltal történő adat lekérdezés ezeken keresztül fog történni. Ezeket az endpointokat egy Map kollekcióban tároljuk , amik kulcsa az endpont címe és értéke végrehajtandó sql lekérdézés és feldolgozott válaszadás lamdba funkciója.

A képen szöveg, képernyőkép, szoftver, Multimédiás szoftver látható

Automatikusan generált leírás

20. ábra: Egy endpoint minta

### /client/

Itt minden a Quasar ( Vue 3-ra épülő ) webes keretrendszer szerinti elrendezést használja , a fő mappánk tool\_analytics\_frontend/src , ahol a forrásfájlok találhatóak. A keretrendszer egy Layout-ra építi a megjelenítést, amiket a navigációknál használt page-k váltanak és azokat felépítő komponensek. Minden \*.vue kiterjesztésű file a megjelenítést szolgálja ami szokásos webes komponenst magába foglal ( html kód , javascript kód és css kód ).

A store mappában a található a pinia-hoz tartozó tartós állapotkezeléshez tartozó kódok.

A boot mappában pedig minden indulás előtti inicializációs modul, mint pl a rest kezelő keretrendszer (axios) , színfeltöltés és segédtábla lekérdezések.

## Futtatási lépések fejlesztőknek:

Alapértelmezetten a helyi gépen a 3306, 8080 és 9000-es portoknak kell elérhetőnek lennie, ha bármelyik ezek közül épp foglalt az gátolja sikeres futtatást. Ezt könnyen megoldhatjuk egy „npx kill-port <portszám>” paranccsal. Szükség esetén újra definiálhatjuk az alapértelmezett portokat is, azonban ilyenkor a kapcsolat mindkét oldalán, illetve a run-script.js-ben is szükséges beiktatni a módosításokat. A port módosításra lehetőségek:

* Adatbázis: docker-compose.yml-ben a ports mező második értéke jelzi a kiszolgáló gépen kapcsolt portot, ilyenkor a változtatás a /server/connection.ts-ben is szükséges.
* Szerver: adatbázis oldalról a fenti eset , kliens oldalról a main.ts portNumber-jét kell átírni. Ilyenkor a client/src/boot/axios.ts-ben az apiBaseUrl visszatérési értékét kell változtatni ezzel együtt.
* Kliens: A quasar.config.js-ben található devServer változó port számát kell átírni.

### Manuális

Fejlesztés közben nincs minden esetben szükségünk arra, hogy a teljes adatbázist újra indítsuk, ha a tegyük fel a szerver kódján dolgozunk. Ilyenkor hasznos lehet manuálisan 3 terminálból külön kezelni a rétegeket.

Változtatás esetén a kliens kódja nem igényel újraindítást, minden változtatás a használt file mentésekor bekerül a futó alkalmazásba. A szerver esetében lekell állítani és újra buildelni majd futtatni a kódot. Az adatbázis esetében a runtime-script-ekben történt változás automatikusan felkerül a container-re. Nekünk csak újra kikell adni a futtatásukhoz használt parancsot. Ha az alaptáblákban történt változás, a docker-compose down paranccsal teljesen lekell állítanunk a futó container-t és újra indítani a docker-compose up-al.

Minden más futtatási lépés és parancs azonos a felhasználó dokumentációban feltüntetettekkel.

### runtime-script.js

## Fejlesztési terv- és folyamatok:

A projekt elkészítését három részre osztottam, mindegyiknek külön git branch-e van, amiket befejezéskor vissza mergeltem a main branch-re:

1. fázis: Tervezés, technológiák kiválasztása, 3 modul alap szerkezetének elkészítése
2. fázis: Modulok összekötése és részletes kidolgozása
3. fázis: Utólagos módosítások, tesztelés, dokumentálás

### Technológiák

|  |  |  |
| --- | --- | --- |
| Adatbázis | Szerver | Kliens |
| * Docker * MySQL | * Express (Node.js) * Express MySQL plugin | * Quasar * Chart.js ( Vue ) * Pinia * Axios |

## Adatbázisrétegek

Az első fontos döntés az relációs adatbázis menedzsment rendszer kiválasztása volt. Azért döntöttem a MySQL mellett mivel egy ingyenes nyílt-forráskódú rendszer és hasonló az egyetemen használt OracleSQL-hez.

Majd az adatbázis réteg tervezésénél fontos szempontnak találtam, hogy a lehetőségek szerint minden lokálisan egy helyről elérhető legyen mindenféle külsős segítség nélkül. Ezért a különböző web service elérést biztosító megoldásokat korán elvetettem (mint pl. AWS cloud ). Azonban egy helyi SQL adatbázis túlságosan helyigényes és nehezen verzió kezelhető lett volna egy git repository számára, ráadásául mi történik ha különböző platformokról használjuk? Korábbi munkahelyemről hozott ötletként merült fel a dokerizált adatbázis elképzelése.

A docker container biztosít egy virtualizált linux környezetben használható MySQL adatbázist, ami elhanyagolható erőforrás igénnyel rendelkezik a helyi számítógépen. Rá adásként biztosított az azonos működése bármilyen platformról használjuk. Így a helyi helytakarékos futtatás és a kereszt-platform elérés is megvalósult, már csak az adatbázis állapotának verziókezelése maradt kérdéses.

Az állapotokat leírhatjuk egy sor sql paranccsal, amiket az adatbázis létrehozásakor lefuttatva megőrizhetjük a perzisztens állapotot és ezeket az sql parancsokat könnyedén verzió kezelhetjük. Már csak ezeket a táblákat és adataikat leíró sql parancsokra volt szükségem.

Az alaptáblák sémáját egy az eredeti mintájáról exportáltam ki a /db/mock/data/00\_schema.sql-ben látható módon. Ez egy sor SQL parancsként reprezentálja a táblákat és azok relációit

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

21. ábra: Sémaleíró szkript részlete

Az abban lévő konkrét adatok viszont nem voltak felhasználhatóak jogilag céges környezeten kívül, ráadásul mennyiségben is jóval több mint amennyi a jelen demonstráláshoz szükséges. A maszkolás hosszú és fölöslegesen munka lett volna ezeken így az adatokat random generáltam egy erre való nyilvános oldalról, majd kisebb átalakításokkal szemléletesebbé tettem. Ezeket a táblájához tartozó névvel elátott \*.sql file-okba helyeztem a /db/mock/data mappában található számsorozott módon.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

22. ábra: Példa egy adat feltöltő script-re

Ezután már csak egy akadály maradt, hogyan fogom elérni, hogy ezek az sql script-ek megfelelő sorrendben végrehajtójának az adatbázis létrehozásakor?

Szerencsémre volt egy beépített megoldás erre az „entrypoint-scripts” formájában. Ez egy előre meghatározott volume amibe a lokális gépről sql file-okat másolhatunk fel, amik végrehajtását biztosítja az adatbázist létrehozó docker script. A kapcsolótábláknak és a tábláknak, amihez kapcsolnak prioritása van létrehozáskor az EventLog táblával szemben, ami hibás végrehajtáshoz vezethetne. Ezért történt a számozás a file-ok között, mivel ez a docker script rendezett sorrendben hajtra végre ezeket az entrypointként felmásolt volume-ban található scripteket.

Ez viszont egy okból nem volt megfelelő a teljes adatbázis tábláinak. Csak automatikusan az adatbázis létrehozása után hajtódott végre. A segédtáblákat pedig szerettem volna, ha dinamikusan futtatási időben is létrehozhatnánk, mivel ezeket direkt módon használja a kliens és így bármikor frissíthettük volna a megjelenített adatokat a teljes adatbázis réteg újraindítása nélkül.

Ezért az ezeket leíró sql parancsok külön egy meghatározott mappába kerülnek fel a container-be, majd manuálisan a MySQL CLI segítségével végrehajtásra kerülnek.

Tehát az adatbázis rétegben szereplő táblák két részre bonthatóak:

* **Első réteg:**A fő relációs adatbázisunk, ezek tartalmaznak minden beérkezett eseményt amiből a kimutatásokat végezzük, plusz a hozzájuk tartozó részletes információkat és azok kapcsolótábláit.  
  A réteg további 3-ra bontható:
  + EventLog tábla: Ez a fő táblánk, itt történik a naplózás minden esemény kiváltásakor.Megtudhatjuk belőle az esemény idejét, milyen parancsot és melyik eszközből kapott. Emelett refernciákat is tárol a felhasználó, lokáció és szoftver adatokról.
  + Információs táblák (User,Location,Software): Ezekben tároljuk a számontartott felhasználókat, lokációkat és szoftver információkat.
  + Kapcsoló táblák: Összekapcsolják a hozzájuk tartozó információs tábla egy elemét egy use session-el(StudioInstance). Ezen párok kerülnek referenciaként egy eseményhez.
* **Második réteg:**   
  A segéd táblák, ezek szolgálnak a kimutatásokhoz tartozó csoportosított lekérdezések eredményeivel és ezek eredménye lesz indirekt módon megjelenítve a diagrammokon.

Segédtáblákra azért volt szükség mivel az éles adatbázis nagyon nagymennyiségű adatot tárol, amik lekérdezése időbe telik és ha ez direkt a kliens kérésére történne az ellehetetlenítene a reszponzivitását.

## Szerver

Az alkalmazás lehetőséget biztosít az adatok dinamikus lekérdezésére a segéd táblákból és kisebb direkt lekérdezésekből. Főbb feladatai:

* **Kommunikáció az adatbázissal: a kapcsolat létrehozása , fenttartása**
* **Adatok feldolgozása: a tábla lekérdezések eredményeinek feldolgozása a megjelenítésnek alkalmas módon**
* **API biztosítása a kliens számára: ahol a feldolgozott adatokat lekérdezhetjük a megjelenítés oldalán**

Emelett session szinten saját magának is tárol segéd táblákat amik a feldolgozást és a címkézést segítik. Pl.: Évek , éppen használt eszközök , stb…

## Kliens

Az alkalmazás egyszerű és felhasználóbarát felhasználói felülettel rendelkezik, amely lehetővé teszi a felhasználók számára az adatok intuitív és könnyű kezelését, valamint a kívánt grafikonok kiválasztását és testreszabását az adatok megfelelő megjelenítéséhez.

## Adatbiztonság és Teljesítmény

Az alkalmazás maximális prioritást ad az adatbiztonságnak és az adatintegritásnak, miközben magas teljesítményt és gyors adatelérést biztosít a háromrétegű adatbázis struktúrájának hatékony kihasználásával.

itt mond h pl bevan egetve a credential a szerverbe

## Összegzés

Az X alkalmazás lehetővé teszi a felhasználók számára, hogy könnyen és hatékonyan navigáljanak a háromrétegű adatbázisban tárolt információk között. A grafikonok változatos megjelenési lehetőségei révén az adatok vizualizációja segíti a felhasználókat a döntéshozatalban és az adatelemzés során.

# Összefoglalás és további fejlesztési lehetőségek

* runtime script logolás debugolás részletesebben
* runtime script paraméterezés
* komponens futtatási paraméterezés : pl portok , mysql jelszavak stb
* mysql adatok beégetve vannak
* több lekérdezés , diagramm
* több összehasonlítás

# Irodalomjegyzék