Tervezői dokumentáció

Adatok tárolását, feldolgozását, megjelentését támogató informatikai rendszer feladat 2. fázisához kapcsolódóan

Tartalomjegyzék

1 A feladat meghatározása

Feladat megnevezése: adatok tárolását, feldolgozását, megjelentését támogató informatikai rendszer

Fázis 1: Az elektroszmog szakterület backend moduljának elkészítése mely megvalósítja az adatok fogadását, tárolását, elemzését, lekérdezhetővé tételét, push értesítést és felhasználókezelést. A fejlesztés során mikroszolgáltatásokra épülő szerver oldali architektúrát alkalmazunk, melynek révén biztosított lesz az egyes szerver oldali komponensek egységbe zárása, skálázhatósága. Adattárolásra open-source relációs és noSQL adatbázisokat alkalmazunk.

Fázis 2: Az elektroszmog tématerület webes kliensének fejlesztése, mely lehetővé teszi az összegyűjtött és elemzett adatok megjelenítését, monitorozást, nyomonkövetést modern webes felületen. A kliens felületen kereshetővé válnak az egyes területek térképes nézetben. Jól használható térképek készülnek az összegyűjtött adatokra alapozottan, melyek szemléltetik az adott területen mért elektroszmog mértékét.

Fázis 3: Az elektroszmog szakterület backend moduljának második változata: elemzés, lekérdezhetővé tétel, push értesítés és felhasználókezelés.

Az elektroszmog tématerület webes kliensének második változata: monitorozás, nyomonkövetés modern webes felületen.

Jelen dokumentum a második fázisban történt kutatási-fejlesztési feladatokat összegzi.

2 Az elkészített szoftverrendszer specifikációja

2.1 A projekt célja

Az Elektroszmog projekt célja az 30Hz – 6GHz között terjedő elektromágneses spektrumban történő térerősség mérések készítése és ezen mérési eredmények tárolása, elemzése és vizualizációja. A dokumentumban tárgyalt projekt az adatok mérőeszközről történő fogadását, feldolgozását, elemzését és vizualizálását tartalmazza. A mérőeszközök és mérési fájlok készítését harmadik fél végzi. A méréseket mobil mérőállomások készítik, az adatok megtekintésére és elemzésére modern webes felületeken lesz lehetőség.

Az elemzések segítségével megállapítható adott pontokban a különböző frekvenciákon mérhető térerősségek alakulása az idő függvényében, vagy adott frekvenciák észlelhetősége pozíciótól függően. Az adatok használhatók az elektromágneses sugárzás egészségügyi hatásainak vizsgálatára, mobiltelefon, rádió és TV műsor szolgáltatók és egyéb rádiós szolgálatok lefedettségének elemzésére, elektromágneses zavarforrások körüli térerősség eloszlás feltérképezésére stb.

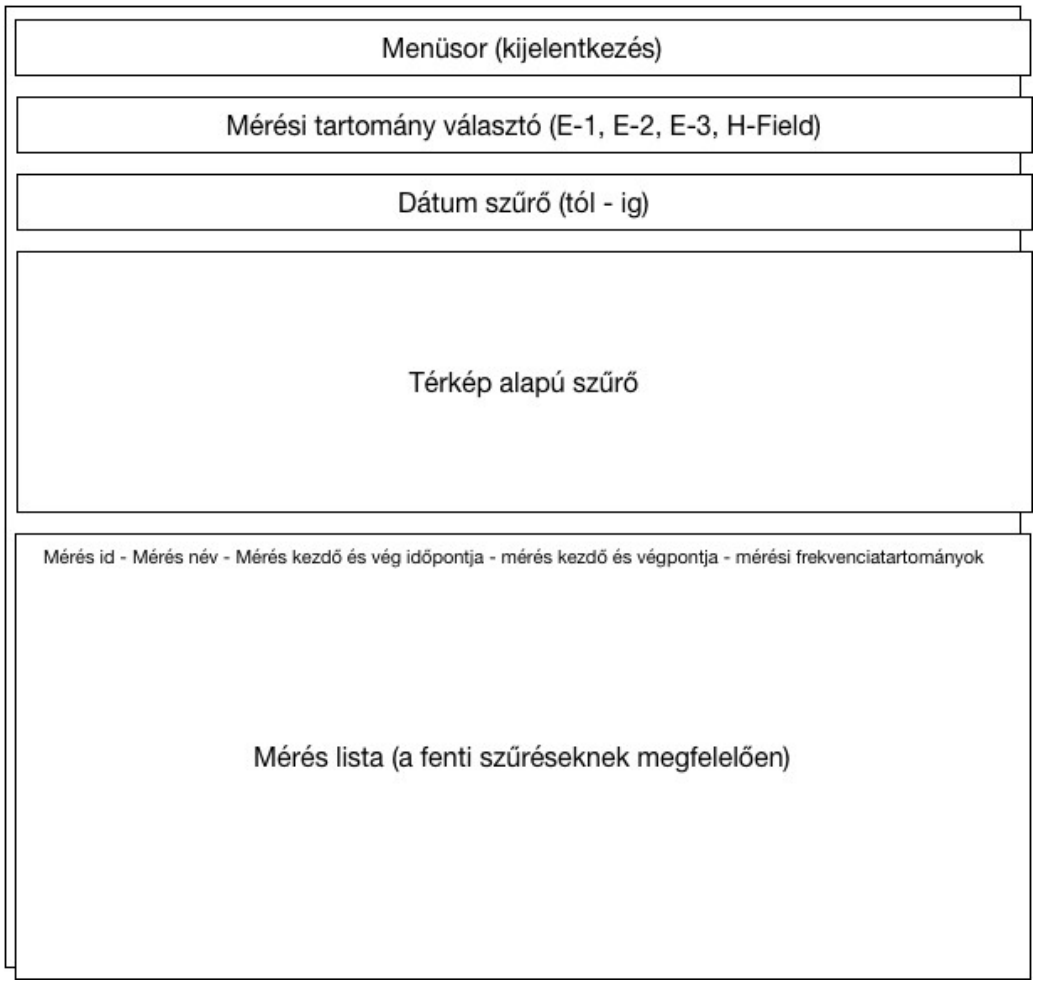
2.2 Követelmények

2.2.1 Megjelenítés

Az alkalmazás modern webes felületen lesz elérhető a felhasználók számára. Az illetéktelen hozzáférést megelőzendő a felhasználónak első lépésben be kell jelentkeznie rendszerbe felhasználónév és jelszó megadásával. Felhasználói regisztrációra az online felület nem ad lehetőséget, az új felhasználók felvitelét az adatbázisba a rendszer üzemeltetőinek kell elvégeznie.

2.2.1.1 Mérés választó felület

Bejelentkezést követően a felhasználó a mérés listázó képernyőt látja.



Mérés választó felület

Felül található egy menüsáv ahol lehetőség van kijelentkezni. Ez alatt ki lehet választani, hogy melyik mérési tartományban készült méréseket szeretné megnézni. A lehetséges opciók:

- H-field 30 Hz - 10 kHz
- E-1 10 kHz - 30 MHz
- E-2 30 MHz - 3 GHz
- E-3 3 GHz - 6 GHz

Emellett lehetőség van a mérések dátum szerinti szűrésére is. Itt egy tól-ig tartományt lehet beállítani. A mérés listában azok a mérések fognak bent maradni, amelyek kezdő vagy végdátuma ebbe a tartományba esik.

Az utolsó szűrési lehetőség térkép alapú, itt más jellegű térkép jelenik meg ha korábban álló, illetve, ha mozgó mérést választottunk ki. Álló mérés esetén a térképen pontszerűen látszanak a mérések helyei. Mozgó mérés esetén a térképen a mérések útvonalának befoglaló poligonjai jelennek meg. Ezzel a megoldással lehetőség lesz kiválasztani több olyan mérést összehasonlításra, amelyek fedik ugyanazokat

a fizikai koordinátákat. A méréseket jelző objektumokra (pontokra vagy poligonokra) történő kattintással az alkalmazás a méréseket vizualizáló képernyőre navigál, ahol a kiválasztott mérés adatait jeleníti meg.

A képernyő legalján egy lista nézet található, ami a szűréseknek megfelelően listázza azon méréseket, amelyek a térképen megjelenhetnek.

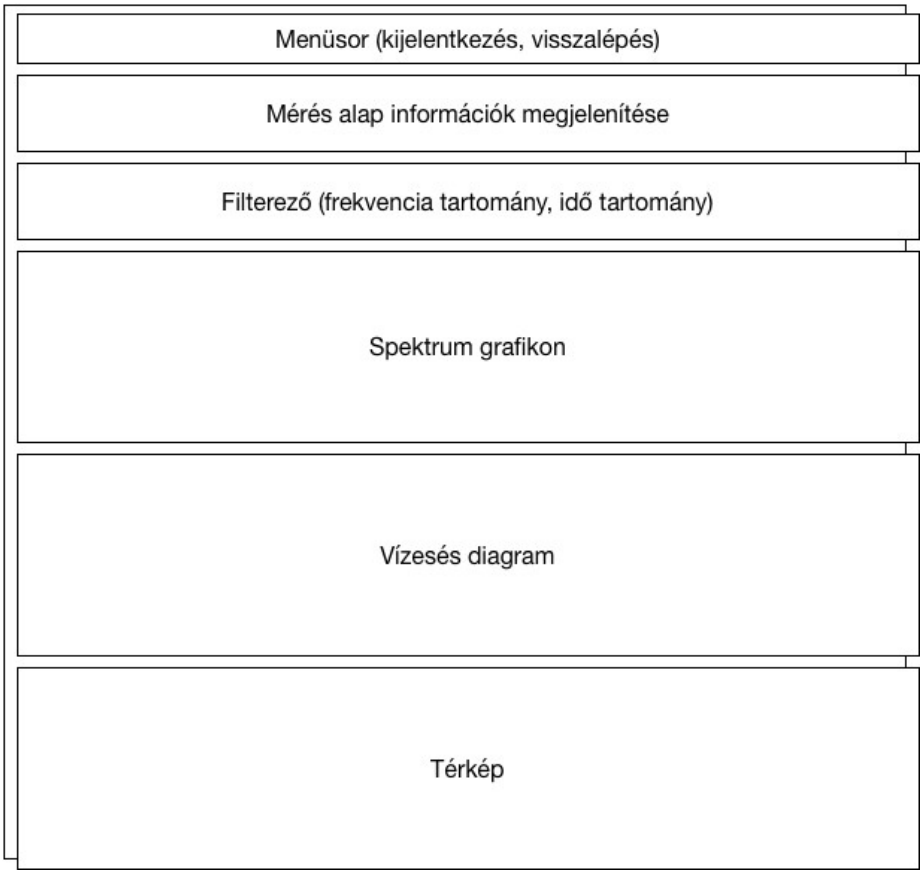
A lista a következő alapadatokat jeleníti meg:

- Mérés azonosító
- Mérés név
- Mérés kezdőpontja (városnév - utca)
- Mérés végpontja (városnév - utca)
- Mérés kezdő időpontja
- Mérés befejező időpontja
- A 4 mérési tartományhoz a mérőberendezésen aktuálisan beállított frekvenciatartomány kezdete és vége

A listából egy vagy több, maximum 3 elem választható ki, majd a megjelenítés gombra kattintva ezek kerülnek vizualizációra.

2.2.1.2 Állóhelyzeti mérések vizualizációja

Megjelenítés szempontjából 2 fő csoportja van a méréseknek, az egyhelyben állva végzettek, ahol a térerősségek időbeli változását szeretnénk vizsgálni és mozgó mérések, ahol a spektrum változásának földrajzi aspektusai érdekesek. Mivel a két esetben más jellegű vizualizációk szükségesek két külön dashboard felületre van szükség.



Állóhelyzeti mérés vizualizáció

A 2. ábrán látható az állóhelyzeti mérések megjelenítésére szolgáló képernyő váza.

A felső menüsoron lehetőség van kijelentkezni az alkalmazásból vagy visszatérni a listázó oldalra.

A menüsor alatt található Mérés alap információk dobozban a listázó képernyőn is látható alapadatok jelennek meg. Ennek a szakasznak a célja, hogy a mérés elemzése során is pontosan tudjuk melyik mérést is látjuk. Amennyiben több mérés lett kiválasztva, mindegyik alapadatai megjelennek.

A filterező lehetőséget ad arra, hogy a vizsgált mérésen belül is tovább szűkítsük a frekvenciatartományt illetve időszakaszt.

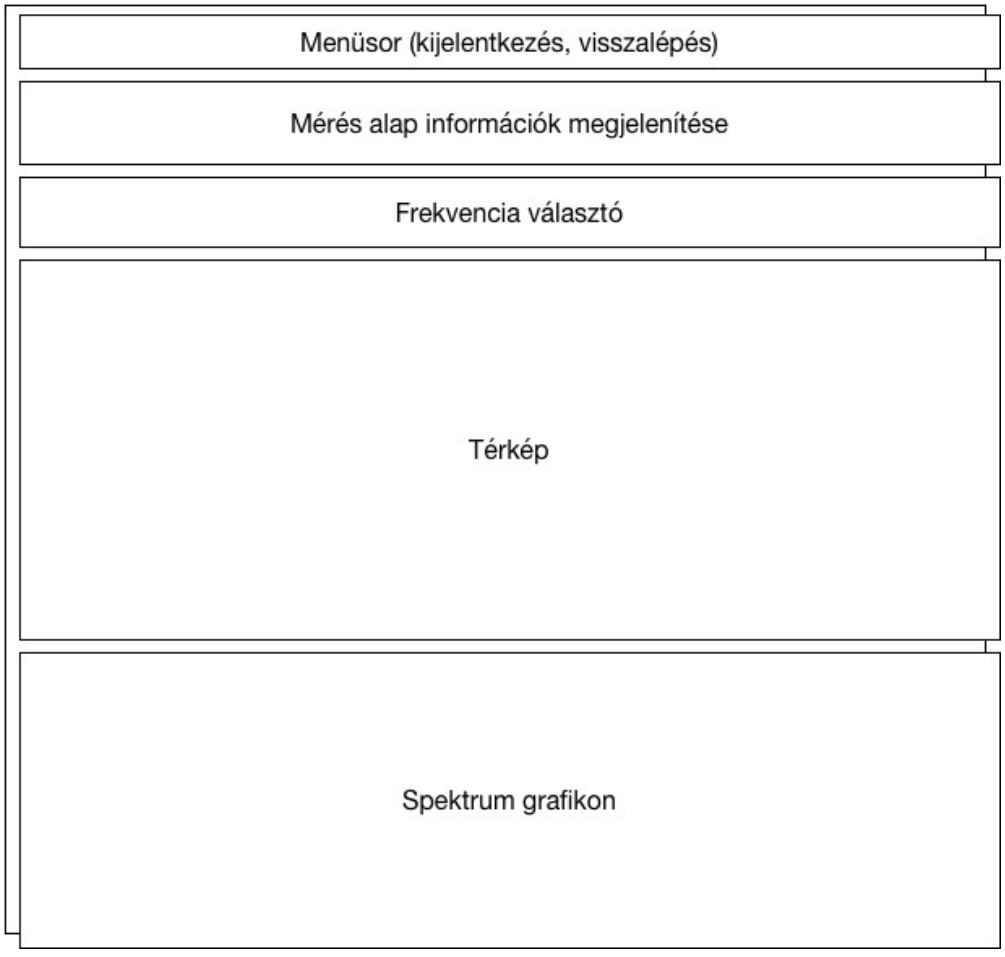
A spektrum grafikonon vonaldiagram formájában lehet megtekinteni a mérési eredményeket. Az X tengelyen a frekvenciák láthatók egységes osztásközzel, 3 tizedesjegy pontosságig. Az Y tengelyen a mért térerősség látható. Mindkét tengelyen jól leolvashatók lesznek a mértékegységek. A grafikonon lehetőség lesz ránagyítani egyes szakaszokra, a grafikon alatti navigációs térkép segít az eligazodásban nagyítás esetén is. A grafikon tooltipet is tartalmaz, az egeret a vonaldiagram fölé mozgatva megjelennek az adott mérési pont koordinátái. Ha több mérést jelöltünk ki, akkor ez a grafikon az összes mérés adatait tartalmazni fogja a könnyű összehasonlíthatóság végett, az egyes adatsorokat különböző színekkel megkülönböztetve.

A vízesés diagramon ez egyes frekvenciák térerősség változásának időbeli lefolyását vizsgálhatjuk. Ez a diagram az X tengelyén a frekvenciákat mutatja a spektrum grafikonhoz hasonló módon, míg az Y tengelyen az idő látható. A grafikon mozog, végtelenítve körbe-körbe jeleníti meg a szivárvány színeivel kódolt térerősségeket. A felhasználónak lehetősége lesz állítani a grafikon mozgási sebességét. Ez a megjelenítési mód csak akkor használható, ha egy mérést választottunk ki. Több kiválasztott mérés esetén inaktív.

A képernyő alján egy térképes megjelenítés is helyet kap, ami egy egyszerű jelölővel megmutatja, hogy pontosan hol történt az adatok rögzítése.

2.2.1.3 Mozgás közben elvégzett mérések vizualizációja

Mozgás közben végzett méréseket az alábbi képernyőn lehet elemezni.



Mozgó mérések megjelenítési felülete

A felső menüsoron lehetőség van kijelentkezni az alkalmazásból vagy visszatérni a listázó oldalra.

A menüsor alatt található Mérés alap információk dobozban a listázó képernyőn is látható alapadatok jelennek meg. Ennek a szakasznak a célja, hogy a mérés elemzése során is pontosan tudjuk melyik mérést is látjuk. Amennyiben több mérés lett kiválasztva, mindegyik alapadatai megjelennek.

A frekvenciaválasztón lehetőség van egy frekvencia kiválasztására. Ezen frekvenciához tartozó mért térerősségek lesznek láthatók a térképes megjelenítésen.

A térképen különböző színekkel és azok árnyalataival kódolva jelennek meg a kiválasztott mérések mérési pontjai, ahol az adott pont színtelítettsége a mért térerősség nagyságával arányos. A mérési pontra kattintva a spektrum grafikonon megjelenik a hozzá tartozó spektrum.

A spektrum grafikonon a térképen kiválasztott mérési pont mért térerősségei látszanak a frekvenciák függvényében. Alapértelmezetten azon mérési pont spektruma látható, ahol a legmagasabb térerősség érték lett mérve. A térképen egy mérési pont kiválasztásával pedig az adott pont spektruma kerül megjelenítésre. A diagram egyszerre több pont spektrumát is képes ábrázolni, ezeket színekkel különbözteti meg.

3 Architektúra, tervezés

3.1 Technológiák

A projekt során egy web alkalmazásnak kellett elkészülni, számos lehetőség állt a kliens oldali felület elkészítésére, de az alkalmazás meglehetősen adatintenzív, így mindenképpen olyan eszközöket kellett választani, ami ezt a tényezőt figyelembe véve is megfelelő teljesítményt tud biztosítani a végfelhasználók számára. Így a felhasználható elemek köre meglehetősen leszűkült, majd a választás egy React alapú megoldásra esett a nagyszerű teljesítménye, könnyed felépítése és a vele elérhető meglehetősen gyors fejlesztési sebesség miatt.

3.1.1 React

A React egy JavaScript könyvtár felhasználói felületek készítésére, utóbbi években nagyon elterjedté vált a remek teljesítménye miatt. Ez főleg az egyirányú adatkötésen és egyszerű felépítésének köszönheti. Így szabad kezet adva a fejlesztőnek, mindenféle extra hozzáadott funkcionalitás nélkül, amely kényelmi szempontból előnyös, de a a teljesítményre hátrányos hatással lenne.

Említésre méltó még a DOM kezelése is, mellyel a megjelenített HTML oldalakat állítja össze. Egy úgynevezett virtuális DOM-ot hoz létre, ezzel a gyakori és lassú DOM módosításokat a memóriában végzi, majd ezt vissza szinkronizálja a böngésző DOM-jába, így csak a valóban módosult elemek kerülnek frissítésre.

Mindemellett megjegyzendő, hogy mivel egy könyvtárról beszélünk és nem egy teljes keretrendszerről, így teljesen szabad kezünk van a további eszközök választásában, így további optimalizálás érhető el a feladathoz illő kiegészítők kiválasztásában. Ezt azonban nehézségként is felróhatjuk számára, mivel elvár egy bizonyos ismeretet a React praktikákról, és a háttértudást az egyes kiegésztő könyvtárak előnyeiről, hátrányairól, használati eseteiről.

3.1.2 Redux

A Redux biztosítja az állapot menedzsmentet az alkalmazásban, ez a méltán népszerű flux tervezési mintán alapul.

A flux egy a Facebook által készített tervezési minta, aminek alapja az adatok menedzselése és az adatok egy irányba való folyásának biztosítása. Ennek köszönhetően a megjelenítési réteg tényleg csak a megjelenítésért felel, az adatok módosításáról pedig egy másik komponens (Action) gondoskodik.

Bár nagyban hasonlít az MVC architektúrára, mégsem az, hiszen itt a feldolgozás egy irányú, nem pedig kétirányú, mint MVC-ben. A Store-ok bármilyen, az alkalmazás állapotához kapcsolódó adatot tárolhatnak, míg MVC-ben a Model csak egy objektum tárolására lett kitalálva. Mivel a kétirányú rendszerekben sokkal nehezebb a hibakezelés is — hiszen nehéz rájönni, hogy éppen hol történt változás a kódban a kaszkádosodás effektusának köszönhetően —, addig flux-ban sokkal könnyebb az események kiváltási helyét megtalálni.

A flux legfontosabb részei kerülnek bemutatásra a következőkben.

3.1.2.1 Dispatcher

Singleton egy alkalmazáson belül. Action-öket kap, majd elosztja azokat a feliratkozott Store-oknak. Minden feliratkozott Store megkap minden Action-t.

3.1.2.2 Store

Az alkalmazás adatait tárolja, beregisztrálják magukat az alkalmazás Dispatcher-ébe, hogy megkapják az adatokat tőle. Az itt található adatokat csak Action-ök segítségével lehet módosítani, emiatt nincsenek setter, csak getter metódusok. A Store-ok saját maguk döntik el, hogy milyen eseményekre válaszoljanak. Minden alkalommal, amikor egy Store állapota megváltozik, egy "change" eseményt kell kiváltani. Egy alkalmazásban több Store-nak is szerepelnie kell.

3.1.2.3 Action

Az Action-ök definiálják az alkalmazás API-ját. Ezek határozzák, meg, hogy milyen módokon lehet az alkalmazással interakcióba lépni. Az action egy egyszerű objektum, rendelekezik egy típus mezővel, de emellett hordozhat további adatot is magában.

Az Action-öknek jól ki kell fejezniük a funkcionalitásukat, de nem kell részletezniük a mögöttes implementációt. Fontos, hogy az egész alkalmazáson belül megkülönböztethetőnek kell lenniük, hiszen minden egyes Store megkapja őket és el kell tudnia dönteni, hogy ez számára érdekes adat-e.

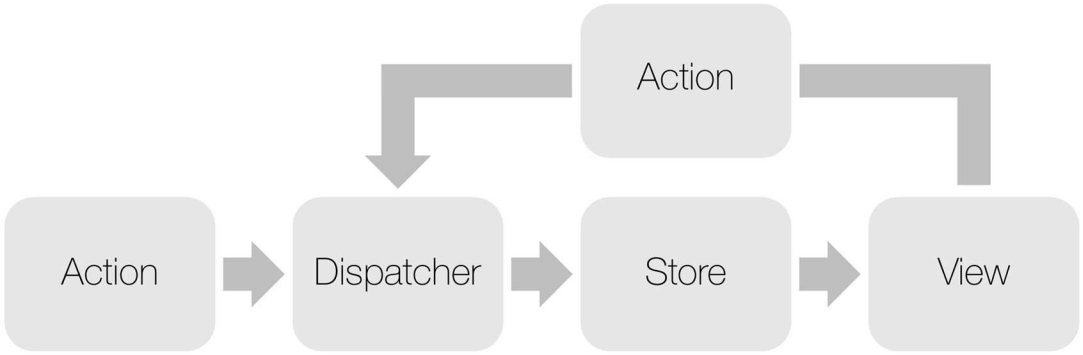
3.1.2.4 View

A Store-okban szereplő adatok jelennek meg itt. A View-k bármilyen keretrendszert használhatnak, mi a React-ot választottuk.

Amikor egy View adatot használ a Store-ból, akkor fel is kell iratkoznia annak a Store- nak minden eseményváltozására, így amikor a Store egy "change" eseményt küld ki, akkor a View megkapja az új adatot és újrarajzolja a felületet az adattól függően. Ez az a rész, ahol az Action-ök általában keletkeznek, hiszen a felhasználó tipikusan az alkalmazás felületével lép interakcióba.

3.1.2.5 Adatfolyam

Az előbb leírt működést, az adatok áramlását az alábbi diagram ábrázolja. (Az ábrán szerepel egy olyan Action is ami a nem a felhasználótól származó eseményeket jelenti.).



Flux adatfolyam

Érdemes megjegyezni, hogy az adatok csak egy irányba mozognak.

3.1.2.6 Alapelvek

A redux a flux tervezési mintára építkezik, három fontos alapelvet követ:

- "Az igazságnak csak egy forrása van"
- Az egész alkalmazás teljes állapota egyetlen Store-ban helyezkedik el
- A State csak olvasható, a módosításának egyetlen módja, hogy Action-t váltsunk ki

3.1.2.7 Miben is különbözik a flux-tól?

A redux-ban nincs Dispatcher, helyette tisztán függvényekre hagyatkozik az esemény kiváltók helyett.

A redux azt feltételezi, hogy adatot nem módosítunk: adatot a reducer-ekben nem módosítunk, helyette egy új objektumot adunk vissza. A dokumentációjából világos, hogy akkor érdemes használni, ha az alkalmazásban nagy mennyiségű adat változik az idő folyamán, illetve, ha fontos, hogy az igazságnak csak egyetlen forrása legyen. Ezeknek köszönhetően nem kell a React-ban mindent egy felsőbb komponensben eltárolni, hanem mindig fordulhatunk az egyetlen adatokat tároló komponenshez, hiszen ott mindig a legújabb, aktuális adatreprezentáció fog szerepelni.

3.1.3 Stíluslapok

A tervezés és fejlesztés során is a mobile-first gondolkodást tartottuk szem előtt, ezt megkönnyítendő a munkánkhoz a Bootstrap komponens könyvtárat választottuk. Ezzel az elkészült alkalmazásunk reszponzív lett, mind asztali gépen mind pedig mobil eszközön kényelmes használatot képes biztosítani.

Az alap konfigurációval nem tudnánk kellően testre szabni a kinézetet, ezért minden további stíluslapot SCSS-ben írunk, ami egy CSS kiegészítő nyelv, mellyel kiforrottabb, stabilabb és professzionálisabb CSS fájlokat írhatunk le.

3.1.4 Adatvizualizáció

A projekt legnagyobb kihívását az adatvizualizáció jelenti, ugyanis több tíz-, néhány helyen pedig több százezer adatpontot kell értelmezhető formában megjeleníteni. Ismerve ennek a nehézségeit, ezen a területen további kutatásokra volt szükségünk. Számos diagram könyvtárat vizsgáltunk meg, hogy melyik fog megfelelni leginkább számunkra. Itt figyelnünk kellett, hogy az elvárt elemek megtalálhatóak legyenek a könyvtárban, vagy a már meglévők bővíthetőek legyenek, hogy megfeleljenek a projekt követelményeiben fogalmazott elvárásoknak.

Több lehetőséget is megvizsgálva a Plotly.js tűnt a legesélyesebbnek, bár React / Redux környezetben több tervezési elvet is sért a felépítése, ezen sajátosságait képesek voltunk komponensekbe zárni és a lehetőségeinkhez mérten optimalizálni. Az így elért teljesítmény az elvárásoknak megfelelt, így a projektben ezen könyvtár mellett döntöttük.

A fejlesztési fázis során teljesítmény problémákra gyanakodva további lehetőségeket vizsgáltunk meg, de az SVG alapú diagrammok között nem tudtunk jobb eredményeket elérni mással. A canvas alapú, valamint WebGL alapú diagram megoldások pedig csak kezdetleges állapotban vannak jellenleg, de további optimalizálás szükségessége esetén ezen a területen további fejlesztéseket elérve lehetne javítani az alkalmazásunk teljesítményét.

3.1.5 További fontosabb könyvtárak

A projekt folyamán számos külső könyvtár felhasználásra került, ezek közül kerül kiemelésre néhány, melyek fontosabb szerepet játszottak az alkalmazás felépítésében.

- Moment: egy pehelysúlyú JavaScript könyvtár, dátumok validálásához, manipulálásához és formázásához. Széles körben elterjedt, segítségével megoldhatóak a dátum kezeléséből adódó nehézségek.
- Axios: széles körben elterjedt Promise alapú HTTP kliens JavaScript környezetben.
- Eslint: kód elemző és hibajelentő eszköz, segít a kódolási hibák elkerülésében, átláthatóbb, könyebben karbantartható kód keletkezik használatával.
- Prettier: kód formázó eszköz, érvényesíti a teljes kódbázisra a formázási szabályokat, ezzel átláthatóbb és egységesebb kód keletkezik.
- Jest: tesztelési környezet, mely tesztek futtatását, ellenőrzését végzi.
- Puppeteer: könyvtár, mely magas szintű hozzáférést biztosít a böngészőhöz, így kontrollált körülmények között futtathatóak az E2E tesztek.
- Enzyme: tesztelési eszköztár, mely megkönnyíti a React komponensek tesztelését.

3.2 A megvalósítandó funkcionalitás

Az alkalmazás követelményeiben megfogalmazott elvárásokban több helyen is ismétlődés, vagy funkcionalitás duplikációja figyelhető meg, így az alkalmazás tervezése során már a kezdetektől figyelünk,

hogy mindent komponensekbe szervezzünk, ezek pedig kellően rugalmasak és paraméterezhetőek legyenek ahhoz, hogy az alkalmazás egy másik pontján újra fel tudjuk őket használni.

A következő fejezetekben az egyes megvalósítandó funkciók tervezői döntései lesznek bemutatva, nehézségeket, kihívásokat kiemelve.

3.2.1 Authentikáció

Az alkalmazás felhasználói egy felhasználónév/jelszó páros megadásával azonosíthatják magukat. Az azonosítás során megszerzett Bearer tokennel minden további kérésünket tudjuk igazolni, így biztosítva, hogy illetéktelen hozzáférés nem történik az adatokhoz.

A kapott tokent tároljuk helyi tárban, így, ha van érvényes hozzáférésünk, akkor a bejelentkezés lépését kihagyhatjuk.

Ezen felület további részletezést vagy tervezést nem igényel.

3.2.2 Mérés kiválasztó felület

A mérés kiválasztó felületen a mérések szűrését kell elvégeznünk, hogy a listázás eredményéből tudjon a felhasználó választani.

Több nehézséget is meg kell oldani. A térkép alapú szűrőhöz a Google Maps JavaScriptes implementációját használjuk, de ezt vezérelnünk kell, hogy az alatta lévő mérési listával szinkronban legyen. Zoomolás, kijelölés stb.

Az egyes mérések kiválasztásához „drag and drop”-os megoldást fogunk alkalmazni, így a mérési lista az egyes szűrések között követhető marad.

3.2.3 Állóhelyzeti mérések vizualizációja

Az egyik legadatintenzívebb felület, melyen számos kontrollnak kell helyet kapnia, valamint a diagrammoknak az időbeni változást is meg kell jeleníteni.

Az egyes diagrammokat sebességre kell optimalizálnunk, hogy adatváltozás esetén amint lehet meg tudja jeleníteni ezt.

A mérés alapinformációk statikus megjelenítésű, az alatta található filterező pedig dinamikusan változtatja a diagrammokat. Az időtartomány vezérlésére pedig egy a már videólejátszókból ismert Lejátszás, Szünet, Megállítás, Ugrás funkciókkal rendelkező vezérlőt készítünk.

A térképes nézeten a megjelenő adat feleslegesen redundáns lenne az alapinformációkban találhatóval, így ebben a pontban eltérünk a követelményekben megfogalmazottaktól.

3.2.4 Mozgás közben elvégzett mérések vizualizációja

A mozgó mérések megjelenítése nagyban hasonlít felépítésben az állóhelyzeti mérések vizualizációjához. A térképen megjelenítjük az egyes mérési pontokat, és azokat az éppen kiválasztott frekvenciáján mért értékük alapján színezzük.

Az egyes mérési pontok kattinthatóak, ezeket a Google Maps JS implementációja biztosítja számunkra, a kiválasztási logikának implementálása és ezek alapján spektrum diagramon való adatmegjelenítés dinamikus változása viszont megoldandó feladat.

Célszerű az itt felhasználandó spektrum diagrammot és a korábban az állóhelyzeti diagrammot azonos komponensel kiváltani, így elkerülve a kódduplikálást és ezen fajta diagramot egységbe zárva, felparaméterezhető és könnyen karbantartható módon kezelni.

4 Megvalósítás

A megvalósítás során nagy figyelmet szenteltünk a komponens alapú felépítésnek, ezzel az alkalmazás felépítése moduláris lett, az egyes komponensek jól elválaszthatóak a többitől, így könnyedén tesztelhetők.

A következő részben az egyes komponensek lesznek részletesebben bemutatva.

4.1 Közös komponensek

Számos komponens több helyen is felhasználásra került az alkalmazásban, ezeknek a bemutatása kerül sor a továbbiakban

4.1.1 Mérés részletező

A komponens alapját egy HTML táblázat képezi, mely mobilos környezetben nem elég rugalmas számunkra, így ezen komponens mobil és asztali környezetben is más-más megjelenítési módot kapott.

Mérés azonosító	abcfdce5-7587-4d47-9a22-d789d3ff568f
Név	example_input
Kezdő pont	Vecsés, Telepi út 32, 2220 Hungary
Befejező pont	Budapest, Ferenc krt. 15, 1094 Hungary
Kezdő időpont	2018. 04. 16. 11:58:20
Befejező időpont	2018. 04. 16. 13:54:06
Frekvencia tartomány kezdete	80MHz
Frekvencia tartomány vége	120MHz
Mérés típusa	Mozgó
Frekvencia tartomány	E-1

Mobil részletező nézet

Míg mobil nézeten egy kártyás megjelenítést használtunk, addig asztali környezetben maradtunk a táblázatos megjelenítésnél.

Mérés Adatok

Mérés azonosító	abcfdce5-7587-4d47-9a22-d789d3ff568f	**Név**	example_input
Kezdő pont	Vecsés, Telepi út 32, 2220 Hungary	**Befejező pont:**	Budapest, Ferenc krt. 15, 1094 Hungary
Kezdő időpont	2018. 04. 16. 11:58:20	**Befejező időpont**	2018. 04. 16. 13:54:06
Frekvencia tartomány kezdete	80MHz	**Frekvencia tartomány vége**	120MHz
Mérés típusa	Mozgó	**Frekvencia tartomány**	E-1

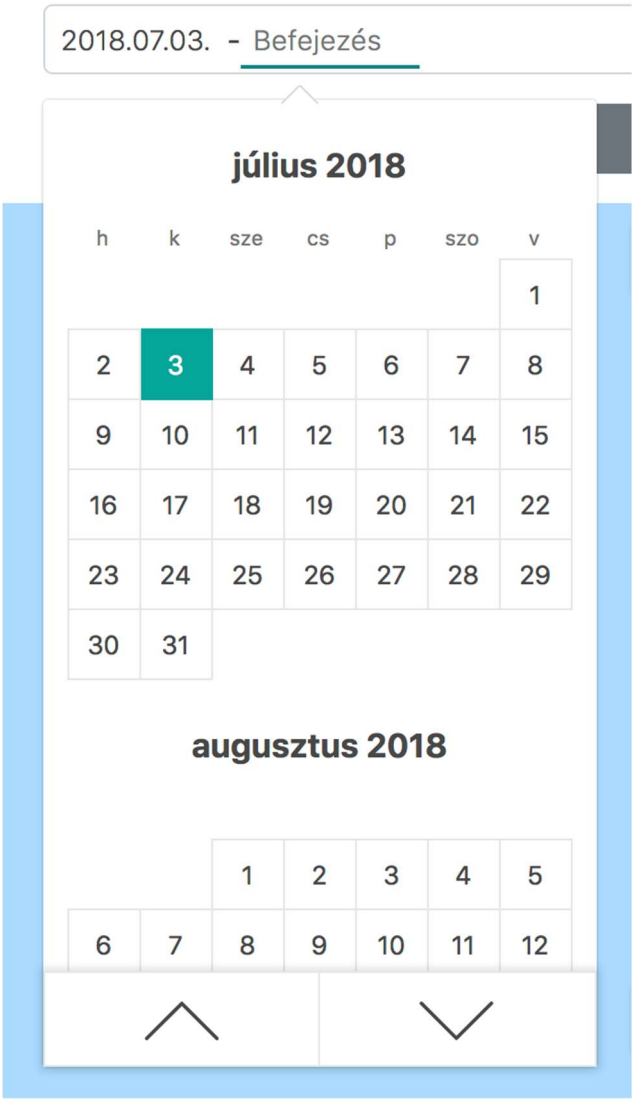
Asztali részletező nézet

Mint látható, az adatok mind a két esetben jól áttekinthetőek.

4.1.2 Idő intervallum választó

Az mérési időtartomány meghatározásához külső megoldást használtunk, mely biztosítja számunkra a kért funkcionalitást és egyben reszponzív megjelenítést is.

A külső elemet becsomagoltuk és kontrollált komponensként tettünk elérhetővé az alkalmazás többi modulja felé, így biztosan egységes lesz a megoldás a teljes kódbázisban, és csak egy helyen használjuk a külső függőséget, igény esetén azonos interfészt biztosítva cserélhető ez a komponens.



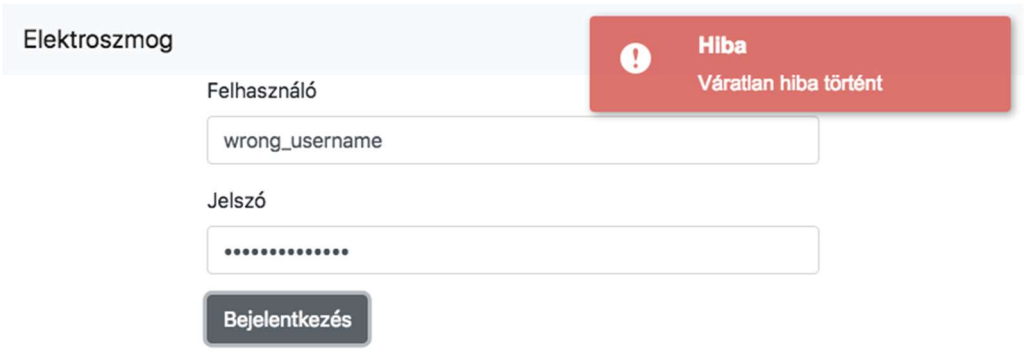
Időintervallum választó

4.1.3 Frekvencia választó

Számos választó komponens készült a fejlesztés során, ezek többségében hasonló elven épülnek fel, a már meglévő HTML selectet alapul véve az egyedi igényekre szabtuk őket. A mérés részletező oldalon látható megjelenésük, a mögöttes logika pedig meglehetősen magától értetődő.

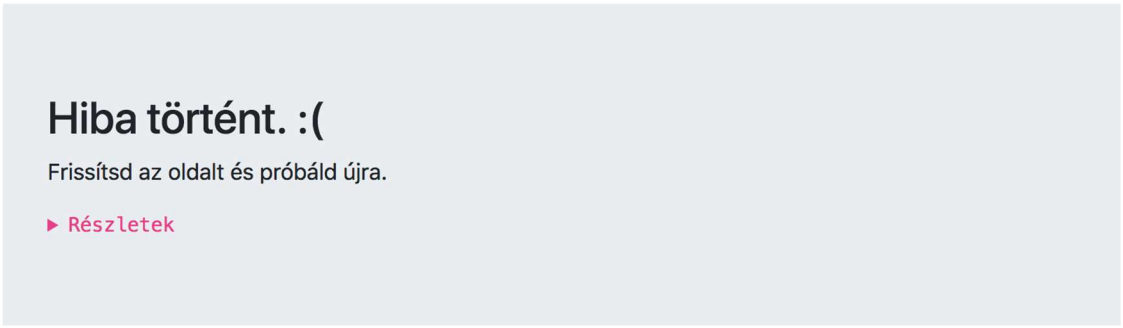
4.1.4 Alkalmazáson belüli hibakezelés

Az alkalmazáson belül több helyen is kezelnünk kellett a hibákat. Ezt megtettük lokálisan, ahol a hiba egy várható esemény. Ilyen esetben egy Toast értesítés figyelmezteti a felhasználót, hogy valami történt, ha a hiba jól behatárolható akkor további információkat is biztosít a felhasználónak a kezelésére.



Toast hibakezelés

Az előre nem látható hibákra is készülnünk kellett, ezek az alkalmazás teljes összetöréséhez vezetnének, ezeket kisebb egységekben is kezelhetjük, de célszerű globálisan az egész alkalmazás alá is elhelyezni egy hibakezelő réteget, ilyen esetben a felhasználó értesítést kap, majd esetünkben megkérjük, hogy az oldal frissítése után próbálja újra.



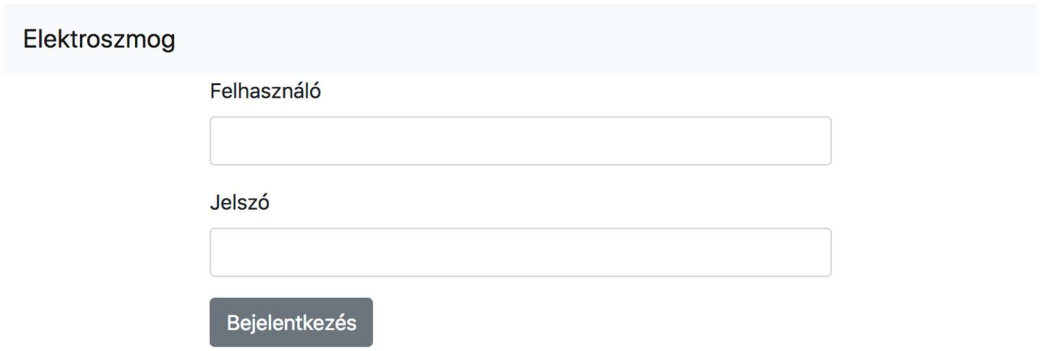
Globális hibakezelés

Fontos megjegyezni, hogy ezen hibák az analitikai rendszernek továbbításra kerülnek, így a felhasználó külön interakciója nélkül is tudomást szerzünk róluk.

4.2 Felületek

A következő részben az elkészült felületek kerülnek bemutatásra

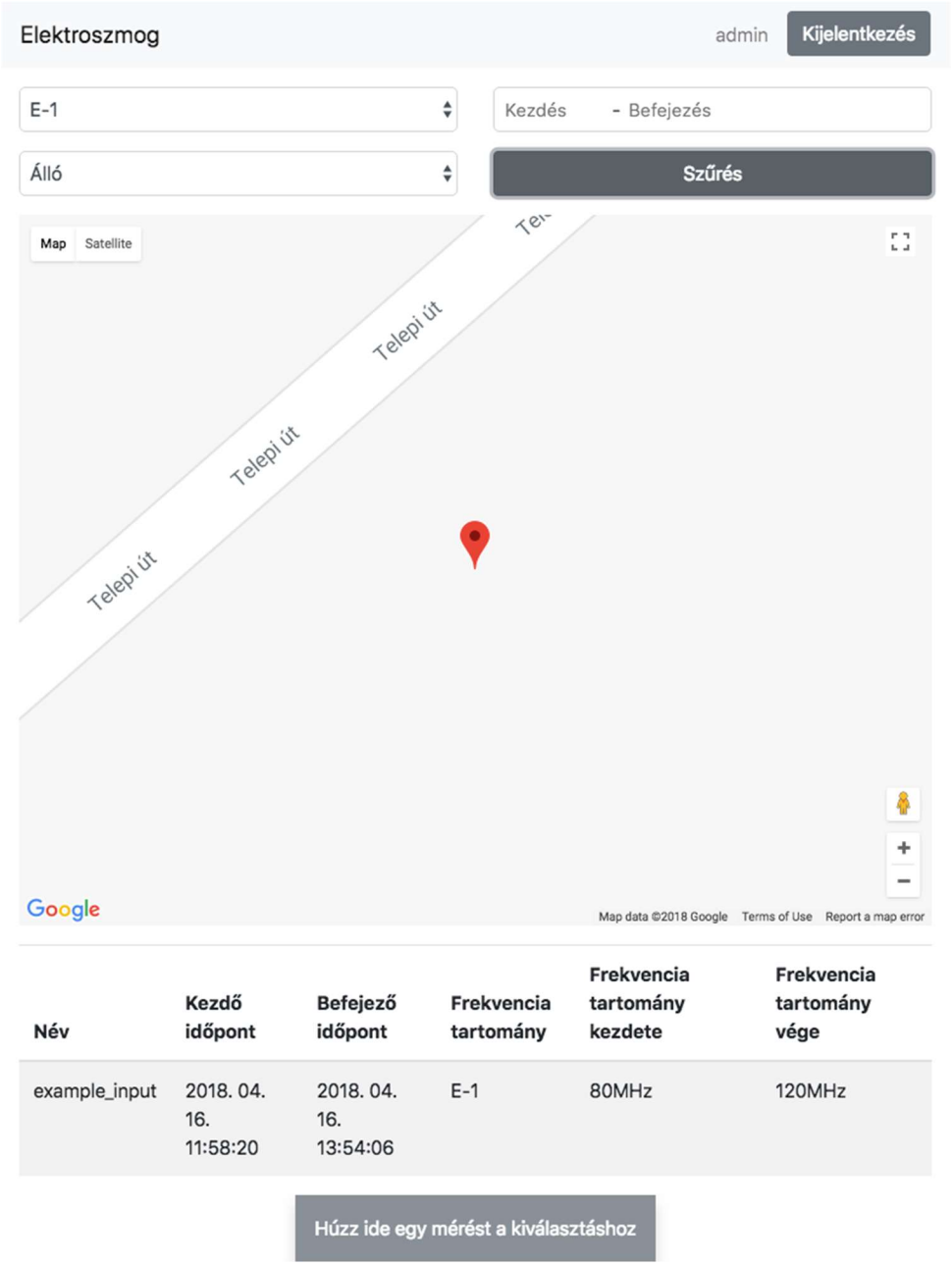
4.2.1 Bejelentkezés



Bejelentkező felület

Minimális kialakítás, csupán a szükséges elemek láthatóak, a korábban már említett hibakezeléssel felvértezve.

4.2.2 Szűrő felület



Szűrő felület

Az alkalmazás nyitóoldala, a felhasználó a fenti filter területen beállíthatja a feltételeket, majd az eredményt a térképen és a táblázatos részen is megtekintheti.

4.2.2.1 Térképes megjelenítés

A térképes megjelenítéshez a Google Maps JavaScript implementációját használtuk fel. Ezen jelenítjük meg a markereket és poligonokat a mérés típusától függően.

A komponens az interakciókat is kezeli, az egeret a térkép felett mozgatva kijelölésre kerül a táblázatos nézetben az mérés sora, valamint ez vissza felé is működik, táblázatos nézetben a térképen kerül kijelölésre a mérés.

A térkép minden szűrésnél kiszámolja, hogy mi lenne neki a megfelelő nagyítás és pozíció, hogy minden eredmény egyszerre látható legyen.

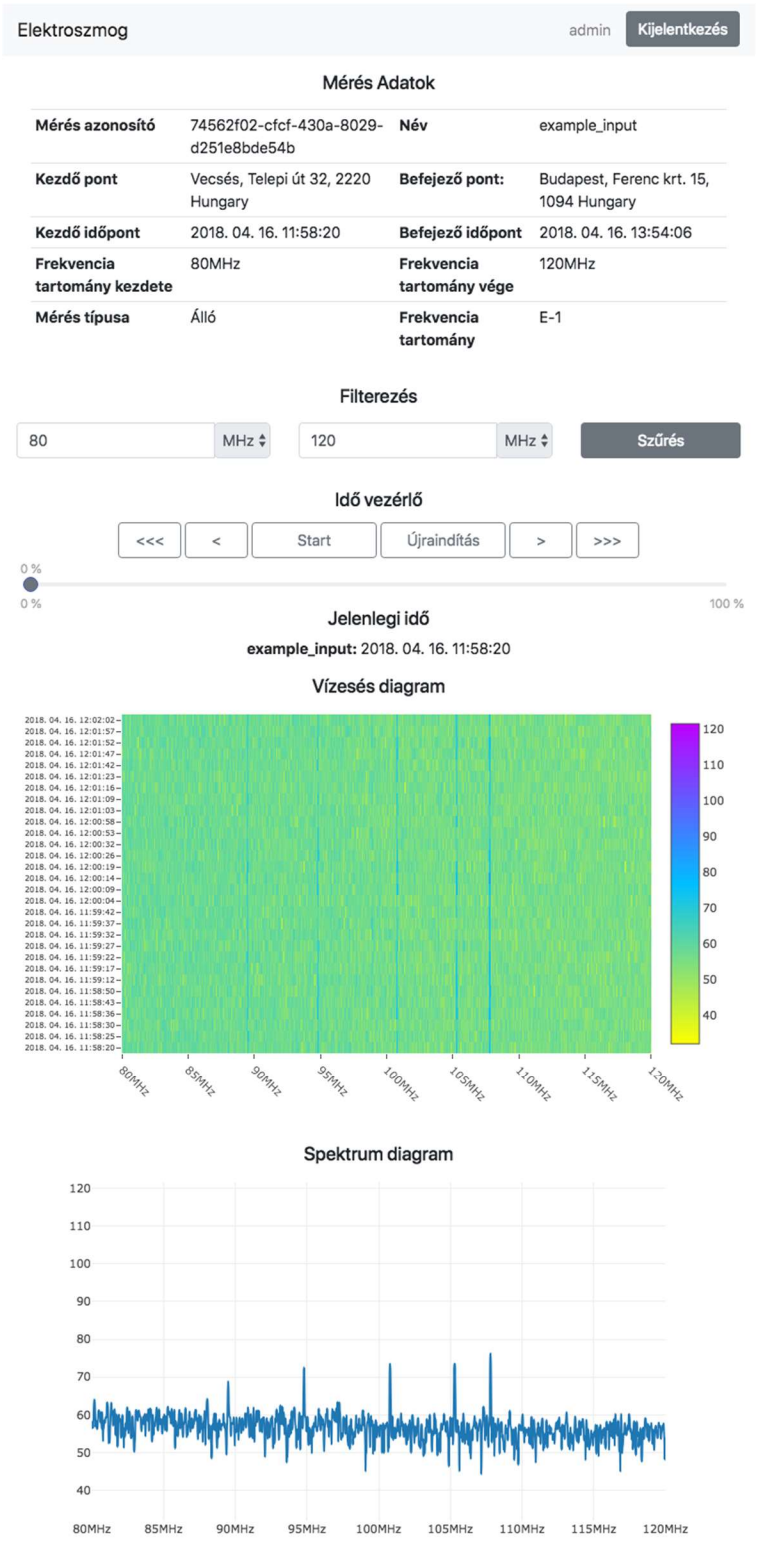
4.2.2.2 Táblázatos megjelenítés

A táblázatos megjelenítés során a fent már említett interakciók és kijelölések kezelése volt a feladat, valamint a kiválasztás lehetőségének megteremtése.

Itt több lehetőségünk is volt, használhattunk volna checkboxos kijelölést, de az szűrések között nem nyújtott volna megfelelő felhasználói élményt, így a „drag and drop” implementálása mellett döntöttünk.

A kiválasztást külön komponens végzik, az egyes sorok a táblázatban pedig rendelkeznek a „drag and drop”-olható tulajdonsággal. A mobilos kompatibilitás miatt más, „dnd” backendet kellett használni, és implementálnunk kellett a dobás közbeni animációt is.

4.2.3 Álló mérések



Álló mérés

A felületre navigálva az fentebbi kép fogad bennünket, a felületet elérhetjük a szűrő oldalról is, valamint URL linkel érkezve is.

Az oldalt betöltve, az alkalmazás a kapott mérési azonosítók alapján a szerverhez fordul, majd az onnan kapott adatokat jeleníti meg.

4.2.3.1 Idő vezérlő

Az idő vezérlő biztosít lehetőséget arra, hogy egy hosszabb mérést teljes időtartamában megtekinthessünk. Erre használhatjuk a gombokat, így navigálva a tartományon, valamint az alatta lévő csúszkán nyomon is követhetjük, hogy merre tartunk az adott mérésen belül. A csúszkát mozgathatjuk is, ezzel teljesen szabad kezet kap a felhasználó a tartományon belüli ugrálásra.

4.2.3.2 Vízesés diagram

A vízesés diagram elkészítése volt az egyik legnehezebb része az alkalmazásnak.

Minden időpillanatban körülbelül 250000 pontot kell megjeleníteni, ezt 3 dimenzióban, miközben, ha az idővezérlővel lejátszásra kapcsolunk, akkor ismételni kell minden időpillanatra körülbelül fél másodpercenként is.

Az első implementáció során nagyon gyorsan elértük a böngésző határait, az SVG alapú megjelenítés annyi DOM elemet módosított, hogy az oldal szinte használhatatlanná vált felhasználói szempontból. Ez köszönhető volt a rengeteg elvégzett számításnak a háttérben.

Rengeteg tervezés és különböző mérések elemzése alapján egy meglehetősen hatékony felépítéssel az adatmanipulációval töltött időt elhanyagolhatóra sikerült csökkenteni, így a teljesítményt szinte teljes egészében a DOM elemek rajzolására és a diagram könyvtár belső számításaira tudjuk használni.

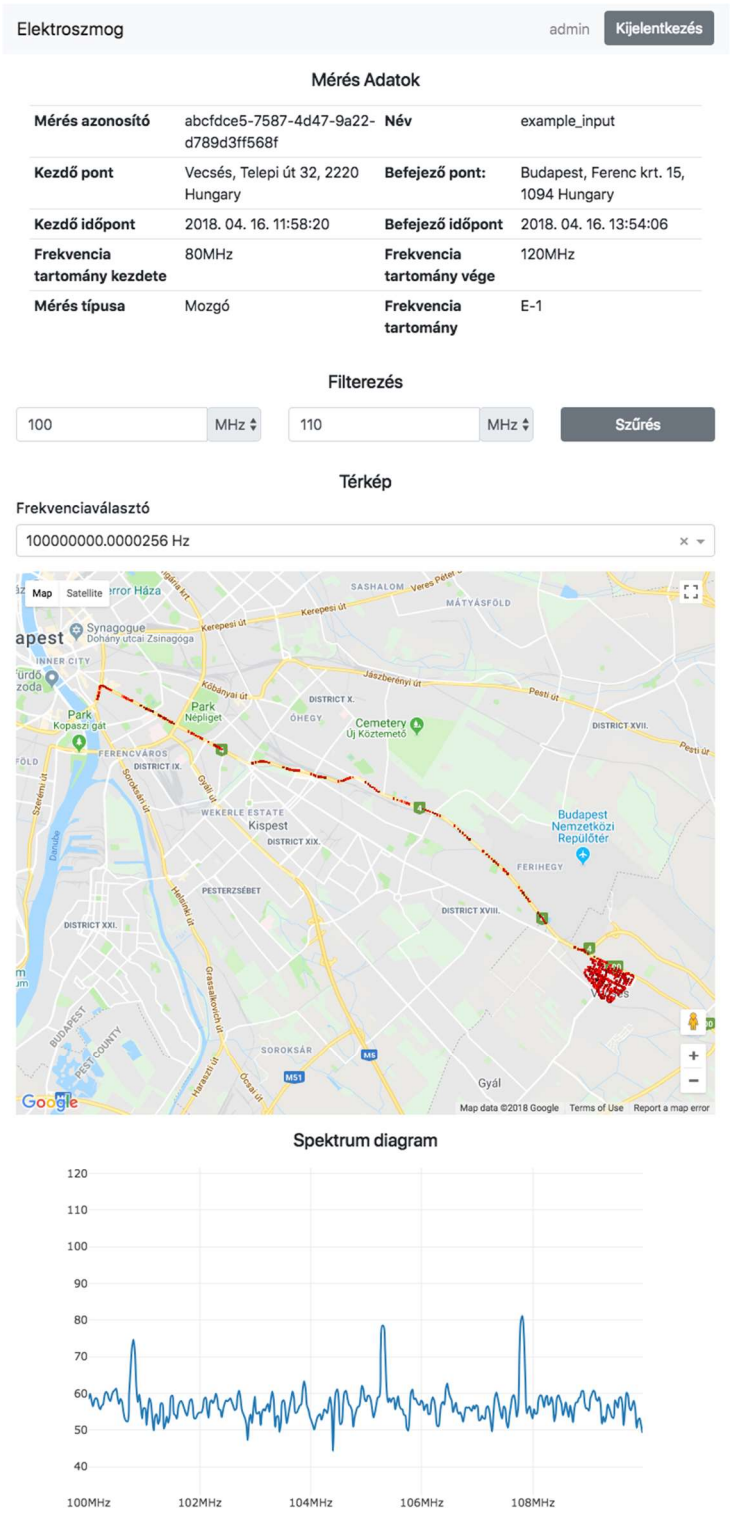
Jól megfigyelhető volt, hogy a React itt sikeresen bizonyította a gyorsaságát, hiszen a redux adatfolyam kialakítása miatt sikerült minimalizálni az általunk végzett számítások időigényét.

További teljesítménynövekedés már csak a Plotly.js cserélésével lenne elérhető, de mivel korábbi elemzéseink azt mutatják, hogy a teljesítménye kiemelkedő az SVG alapú könyvtárak között, a következő lépés már valamilyen WebGL alapú megoldás bekötése lenne. Szerencsére a reactos kódszerevezés remek lehetőséget ad számunkra a továbbfejlesztése. Azonos interfészt biztosítva azonnal cserélhető a megjelenítés.

4.2.3.3 Spektrum diagram

A spektrum diagram már egy meglévő komponensre épít, jelen esetben is becsomagoltuk és személyre szabtuk az interfészét, így könnyedén tudtuk több helyen is használni az alkalmazáson belül.

4.2.4 Mozgó mérések



Mozgó mérések

A felületre navigálva az fentebbi kép fogad bennünket, a felületet elérhetjük a szűrő oldalról is, valamint URL linkel érkezve is.

Az oldalt betöltve, az alkalmazás a kapott mérési azonosítók alapján a szerverhez fordul, majd az onnan kapott adatokat jeleníti meg.

4.2.4.1 Hőtérkép

A hőtérkép meglehetősen összetett feladat volt, a Google Maps JS implementációja a megjelenítést biztosította, utána viszont elengedte a kezünket, és a pontok kijelölését, ezek térképen való megkülönböztethetőségét, színét már nekünk kellett implementálni.

A kijelölés tartogatott néhány kihívást, React sajátossága, hogy ha nem érzékeli egy gyermek komponens változását akkor optimalizálási okokból nem rajzolódik újra, emiatt minden kijelöléskor kézzel kellett triggerelni a gyermek komponens újra rajzolását is, ezt a gyermek komponens referenciájának módosításával érhetjük el.

Mivel a komponens fentről van vezérelve ezért minden kijelöléskor vissza kell szólnunk a szülőnek, hogy mi lett kijelölve.

4.2.4.2 Spektrum diagram

Az álló méréseknél használt diagrammot könnyedén fel tudtuk használni itt is. A térképen való jelölgetés ad hozzá vagy vesz el a spektrum diagrammon megjelent adatsorokból, ezért volt fontos a térkép fentről vezérelhetősége.

4.3 Analitika

Az alkalmazásba beépítésre került a Google Analytics, minden adatot anonim módon rögzítünk, semmilyen módon nem kapcsolhatóak semmilyen felhasználóhoz.

4.3.1 Felhasználó interakciók

Rögzítésre kerülnek a felhasználók interakciói, ezzel megfigyelhetjük melyek a népszerű funkciók, érdemes volt-e gyorsgombot helyezni a szűrő felületre stb.

Jellemzően a gombok, inputok, és más felhasználó interakciók lettek megjelölve. Ezek témakörönként csoportosításra kerültek, hogy az elemzéskor ezekkel könnyítsük a szűréseket.

Az adatok a GA rendszerében tovább aggregálhatóak és további hasznos UI és UX következtetések vonhatóak le belőlük.

4.3.2 Hibajelentés

Az analitika alá helyeztük a hibajelentések küldését is. Számtalan tanulmány bizonyította, hogy az alkalmazások hibái csak sokára jutnak el a fejlesztőkhöz, nem értesülnek róluk megfelelő részleteséggel. Mi ennek elébe mentünk, és a korábban bemutatott hibakezelésbe beépítettünk egy reporting

szolgáltatást is, mely bármely nem várt hiba esetén ezt GA felé feltölti. Majd a webes felületen ez is tekinthető, a teljes hibát kiváltó előzménnyel együtt.

4.4 Tesztelés

A projekt folyamán fontos szerepet kapott a tesztelés, a főbb funkcionalitások, melyek nélkül az alkalmazás életképtelen lenne, teljes mértékben lefedésre kerültek a tesztekkel. Ez a fejlesztés során is előnyösnek bizonyult, hiszen sokkal bátrabban tudtunk 1-1 funkcióhoz hozzányúlni átdolgozási céllal.

A Jest támogat egy úgynevezett snapshot tesztelési módot, mely a tesztelési folyamatot és tesztek írását teszi kényelmessé a fejlesztők számára. Mi a tesztek többségét erre építettük, így tartva fent az érdeklődést és a kreativitást a fejlesztőkben, a sokak által unalmasnak vagy haszontalanabbnak tartott tesztek írásakor.

Ezeket szokás regressziós teszteknek is nevezni, ugyanis nem előre definiált a teszt kimenetele, hanem egy jónak gondolt állapotot mentünk el, és minden újabb állapotot a korábban elmentetthez hasonlítunk.

4.4.1 Unit tesztelés

A unit tesztelést tekinthetjük a szoftverfejlesztés kötelező velejárójának, ez webes környezetben sincs máshogyan, bármilyen logikát tesztelni kell. Ez a tendencia a modern webes környezetben csak még jobban előtérbe került.

Mi is fontosnak éreztük, ezért minden fontosabb elemre elkészült a unit teszt, emellett a triviális megoldásokat nem teszteltük, de redux flow minden pontjára készítettünk mintatesztet, ezzel biztosítva egy remek kiindulási alapot a továbbfejlesztésére.

A unit teszteket bekötöttük a CI rendszerbe, így unit tesztet sikertelenül teljesítő kódrészlet nem kerülhet a fő ágakra, így productionbe sem.

4.4.2 Integrációs tesztelés

A komponens alapú felépítésből adódik, hogy a komponensek együttműködését is tesztelnünk kell, erre is tökéletesen alkalmasak a snapshot tesztek, az enzym által nyújtott tesztelési segédfüggvények különböző mélységekbe tudnak renderelni, így a kirajzolt HTML kódok kerülnek be a snapshotokba és ezzel lesznek összehasonlítva.

Az integrációs tesztjeink továbbra is elszigetelten futnak, és CI rendszerbe vannak kötve, akárcsak a unit tesztjeink. Fejlesztés közben a unit és integrációs teszteket nem különböztettük meg különösebben, hiszen mind a kettő gyorsan fut le esetünkben, így unit tesztként futnak az integrációs tesztek is.

4.4.3 E2E tesztelés

Ezen tesztelési forma ritkábban fordul elő az alkalmazásokban, de ez biztosítja, hogy valóban működő alkalmazás kerül a felhasználó elé. Kiválthatjuk vele a kézi „checklist” tesztelést.

Esetünkben a funkcionalitás meglehetősen véges halmazú, így könnyedén írhattunk minden főbb funkcióra E2E tesztet.

Az tesztek során az egyes pillanatokról képernyőképek készülnek, és ezek kerülnek összehasonlításra a korábbi eredményekkel.

Fontos megjegyezni, hogy ezek a képek eltérhetnek különböző platformok között. Minden esetben chromiumban futnak a tesztjeink, de különböző platformok más és más módon implementálták a betűk renderelését, így kisebb eltéréseket lehet felfedezni, ami képes „false negative” teszteseteket mutatni.

5 Üzemeltetési leírás

Alkalmazásunk üzemeltetésére két lehetőségünk van.

Használhatjuk a hagyományos Node.js környezetben történő fordítást majd static HTML oldal hosztolását egy erre alkalmas webszerveren, ilyen esetben az alkalmazás a helyi szerver /api útvonalán próbálja elérni a backendet. Ezt az adott webszeren biztosítani kell számára, vagy proxy-t beállítanunk erre az útvonalra.

Emellett lehetőség van az elkészített Dockerfile használatára is, ebben az esetben, az alkalmazásunk egy Node.js Docker containeren belül lefordul, majd a build eredményét egy nginx dockerbe másolva biztosít egy bekonfigurált webservert a futtatáshoz.

5.1 Rendszerkövetelmények

A két eset két különböző rendszerkövetelményt fogalmaz meg.

- Static HTML kiszolgálására alkalmas webszerver, valamint a project fordításához egy Node.js környezet.
- Docker környezet

5.2 Szoftverkomponensek telepítése

5.2.1 Webszerver esetén

- Projekt lefordítása: `npm run build`
- A build mappa tartalmának a kiszolgáló szerverre másolása
- Proxy szerver beállítása

5.2.2 Docker esetén

- Buildelés: `docker build .`
- Az elkészült docker image elindítása