T083676/2021.

SZAKDOLGOZAT

KOVÁCS GÁBOR TAMÁS

2021Pécsi Tudományegyetem

Műszaki és Informatikai Kar

Mérnökinformatikus Szak

**SZAKDOLGOZAT**

Hálózati log monitorozó alkalmazás

Készítette: Kovács Gábor Tamás

Témavezető: Laborci Gergely

Pécs

### 2021

### PÉCSI TUDOMÁNYEGYETEM

**Műszaki és INFORMATIKAI Kar**

**Mérnökinformatikus Szak**

**Szakdolgozat száma:**

**T083676/2021.**

**SZAKDOLGOZAT FELADAT**

**Kovács Gábor Tamás**

**hallgató részére**

A záróvizsgát megelőzően szakdolgozatot kell benyújtania, amelynek témáját és feladatait az alábbiak szerint határozom meg:

**Téma:** Hálózati log monitorozó alkalmazás

**Feladat:**  A feladat egy desktop alkalmazás készítése JavaScript nyelven, Cordova vagy Cordova alapú keretrendszerben, egy választott JavaScript (typescript) framework felhasználásával. Az alkalmazás TCP-n vagy (Unix)Socket-en hallgatózik, és képes az ott érkező adatcsomagok megjelenítésére, listázására függetlenül attól, hogy pontosan milyen jellegű adat érkezik. Az alkalmazásba modulként különböző megjelenítők lesznek illeszthetőek, melyek, ha illeszkednek az üzenet mintájára, akkor speciális módon jeleníthetik meg az adott adatcsomagot (pl kép, SQL utasítás, Exception trace, stb).

A szakdolgozat készítéséért felelős tanszék: Rendszer- és Szoftvertechnológiai Tanszék

Külső konzulens: …………………………………………

munkahelye: ………………………………………….

Témavezető: Laborci Gergely

munkahelye: Pécsi Tudományegyetem

Pécs, 2020. október 2.

Dr. Iványi Péter

szakvezető

HALLGATÓI NYILATKOZAT

Alulírott szigorló hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye. A felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Egyéb jelentősebb segítséget nem vettem igénybe.

Az elkészült szakdolgozatban talált eredményeket a feladatot kiíró intézmény saját céljaira térítés nélkül felhasználhatja.

Pécs, 2020. május 29.

|  |  |
| --- | --- |
|  | .................................................  hallgató aláírása |

Köszönetnyilvánítás

Szeretném megköszönni a tanáraimnak a kitartását és szakmai tudását, amellyel segítették egyetemi tanulmányimat és fejlődésemet. Külön szeretném megköszönni a családomnak a támogatását, amely nélkül most nem tartanék ott, ahol tartok.

**Tartalomjegyzék**

**1. Bevezetés**

A dolgozat célja, hogy egy olyan desktop alkalmazást hozzak létre, amellyel monitorozni lehet hálózaton érkező adatokat. Részletesebben, TCP kapcsolattal kapunk adatcsomagokat, médiafájlokat, fájlokat és ezeket listaszerűen megjelenítjük attól függően, hogy milyen adatot kaptunk.

A dolgozat arra ad megoldást, hogy kiszolgálókat, webszervereket, webalkalmazásokat tudjunk felügyelni és ellenőrizni távolról vagy az adott kiszolgálóról.

A célja a dolgozatnak, hogy alternatív megoldást nyújtson a sok neten fellelhető hálózati monitorozó alkalmazás mellett, melyek minden hálózati adatot monitoroznak, nem csupán a nekünk szükséges adott porton beérkező adatokat dolgozzák fel és nem a számunkra jelen esetben fontos adatok megjelenítésére szolgálnak.

**2. Megoldandó probléma megfogalmazása**

A feladat megoldásához szükséges szerver-oldali alkalmazás elkészítéséhez szükséges egy kliens alkalmazás elkészítése is, amellyel adatcsomagokat tudunk küldeni a szerver alkalmazásnak egy tetszőleges TCP porton.

A szerver-oldali alkalmazásban meg kell valósítani, hogy az előbb említett TCP hálózati porton hallgatózzon, majd, ha ezen a porton érkezik egy adatcsomag azt megfelelően annak, hogy milyen típusú az adatcsomag feldolgozzuk. Ezek után fel kell dolgozni az adatcsomagban található adatot az adat típusa szerint is, hogy egy SQL parancs az üzenet, egy ERROR vagy egy kép. Ha szöveges üzenet, amit logolnunk kell azt a típusának megfelelő színnel kiírjuk egy időbélyeggel ellátva, ha pedig egy fájlt kapunk meg kell néznünk, hogy milyen típusú és annak megfelelően megjelenítjük egy kép, video vagy esetleg audio tag-el és szintén időbélyeggel ellátva.

Megoldandó probléma még, hogy könnyen bővíthetőnek kell lennie az alkalmazásnak utólag különböző megjelenítő modulokkal. Ezeknek a moduloknak a célja, hogy különböző típusú adatokat dolgozzanak fel, amelyek még nincsenek kezelve.

**3. Megoldási módszerek elemzése**

Ebben a fejezetben a különböző lehetséges megoldási módszereket fogom leírni, összehasonlítani. A fejezetben részletezni fogok kettő, a feladatban szereplő hálózati kapcsolati megoldást (TCP és Unix Domain Socket) valamint a lehetséges webes alapú keretrendszereket, amelyek közül választottam. Megemlítem az adatok kódolására szánt megoldásokat, amelyeknek a feladatomban hasznát veszem.

**3.1. Hálózati megoldások**

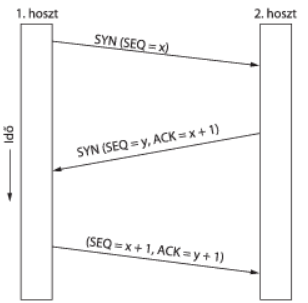
A következő alfejezetekben a lehetséges hálózati megoldásokat fogom elemezni, majd az összehasonlítás után érvekkel alátámasztva kiválasztom a véleményem szerinti megfelelőbb megoldást.

**3.1.1. Unix Domain Socket**

A Unix Domain Socket egy IPC (inter-process communication) megoldás, amely egy olyan kommunikáció, amely két azonos hoszton futó folyamat közötti „hálózati” kapcsolatot valósít meg. Hálózati címek helyett elérési útvonalhoz van kötve a fájlrendszerben, mivel a socketet egy a fájlrendszerünkben látható fájlként hozza létre. A Unix Domain Socket tudja, hogy ugyanazon a rendszeren fut le, ahol a másik folyamat, amivel kommunikál, ezért nem számol checksum-ot, nincsenek fejlécek a csomagokban és útválasztásra sincs szüksége (routing), mindezért nagyon gyors a kommunikáció. Hátránya, hogy távoli kapcsolat nem létrehozható, ezért nem lehet különböző hosztok között hálózati kapcsolatot létrehozni a használatával. [1]

**3.1.2. TCP**

A TCP (Transmission Control Protocol – átvitelvezérlő protokoll) egy végpontok közötti megbízható bájtfolyamot biztosító protokoll. A TCP-t különböző tulajdonságokkal rendelkező összekapcsolt hálózatokhoz tervezték azért, hogy alkalmazkodva ezekhez a tulajdonságokhoz, összekösse őket. A TCP kapcsolat úgy jön létre, hogy mind a küldő és mind a fogadó fél létrehoz egy socketet mint végpontot. Minden socketnek van egy címe, ami a hoszt IP-címéből és a port azonosítójából tevődik össze. Ezekre a címekre az adatfolyamot a TCP maximum 64KB-os darabokba szétszedve küldi tovább melyeket a kézbesítés után újra egy bájtfolyammá alakít. Az összeköttetés létrehozásához úgynevezett „háromutas kézfogást” alkalmaznak, melyet az alábbi ábra szemléltet.



1. ábra, TCP-összeköttetés létesítése, háromutas kézfogás [2]

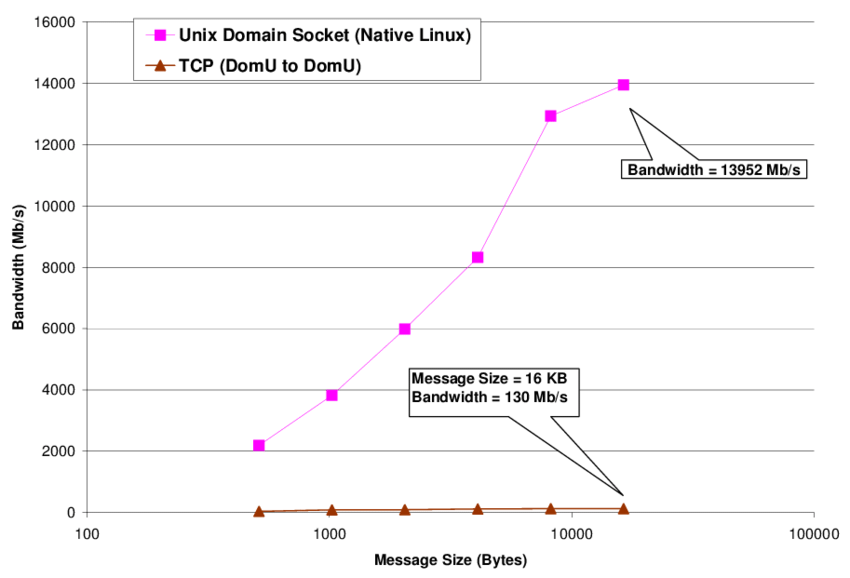
A háromutas kézfogás annyit jelent, hogy időbe telik a kapcsolat létrehozása, amivel is később kapjuk meg az adatcsomagunkat, ami egy hátránya a TCP-nek jelen esetünkben. Az előnye, hogy távoli hosztokkal is lehet kapcsolatot létesíteni a socketek címének köszönhetően. Abban az esetben, ha ugyanazon a hoszton belül szeretnénk folyamatok között használni, akkor a TCP egy loopback interfészt hoz létre, amihez ugyanúgy szükséges a háromutas kézfogás a kapcsolat létrejöttéhez, így majdnem ugyanolyan sebességgel tudunk azonos hoszton futó folyamatok közötti kapcsolatot létrehozni, mint távoli hosztokon futók között. TCP használata esetében vigyáznunk kell arra, hogy ne használjunk olyan portot amely már használatban van más folyamat által. [2]

**3.1.3. Összehasonlítás, választás**

Mind a kettő megoldásnak vannak remek előnyei és kisebb-nagyobb hátrányai.

A Unix Domain Socketnek nagy előnye a gyorsasága (2. ábra), viszont hátránya, hogy másik hoszttal (jelen esetünkben a webalkalmazással, webszerverrel stb.) nem lehet kapcsolatot létrehozni távolról. Ezt a megoldást akkor lehet alkalmazni, hogyha például a webszerverünket magán a kiszolgálón akarjuk monitorozni.

A TCP kapcsolatnak a sebessége a hátránya a Unix Domain Sockettel szemben (2. ábra), és a távoli kapcsolat létrehozása pedig az előnye. Az előbb említettek végett ezt a megoldást főleg távoli kiszolgálók monitorozásához érdemes használni és abban az esetben, hogyha az elhelyezkedésünk változik, mivel csak a hálózati címünket kell megváltoztatnunk és máris működik az alkalmazásunk más helyről, hálózatról is.



2. ábra, TCP és Unix Domain Socket sávszélességek összehasonlítása [3]

Az előbb említett hátrányok és előnyök miatt a TCP kapcsolatot választottam, mert véleményem szerint - mivel az alkalmazásunk nagyrészt kisebb adatcsomagokat kap - nem feltétlen szükséges a szélesebb sávú kapcsolat és így távolról is lehet monitorozni a számunkra felügyelni szánt kiszolgálót.

**3.2. Lehetséges keretrendszerek, környezetek**

Ebben az alfejezetekben leírom és elemzem a lehetséges keretrendszereket, környezeteket, amelyek megfelelnek a feladatkiírásban leírtaknak. Megemlítek egy back-end környezetet is, amely hasznos lehet a későbbi munkámban. A feladatom Cordova, Cordova alapú vagy egy Javascript (typescript) alapú keretrendszer használatával a monitorozó alkalmazás elkészítése. Ezeket a keretrendszereket a feladatomhoz fontos és szükséges tényezőiknél fogva összehasonlítom és kiválasztom a feladathoz legmegfelelőbb keretrendszert.

**3.2.1. Node.js**

A Node.js egy nyílt forráskódú, back-end, JavaScript alapú környezet, amely a böngészőn kívül futtatja a JavaScript kódot. A Node.js lehetővé teszi a fejlesztők számára a JavaScript használatát parancssori eszközök írására és szerver-oldali parancsfájlok készítésére. A Node.js olyan modulokat is biztosít számunkra, amelyek megkönnyítik webalkalmazások fejlesztését. Ilyen modul például a net modul, melynek használatával TCP hálózati kapcsolatot lehet nyitni szerver és kliens között. [4]

**3.2.2. Angular**

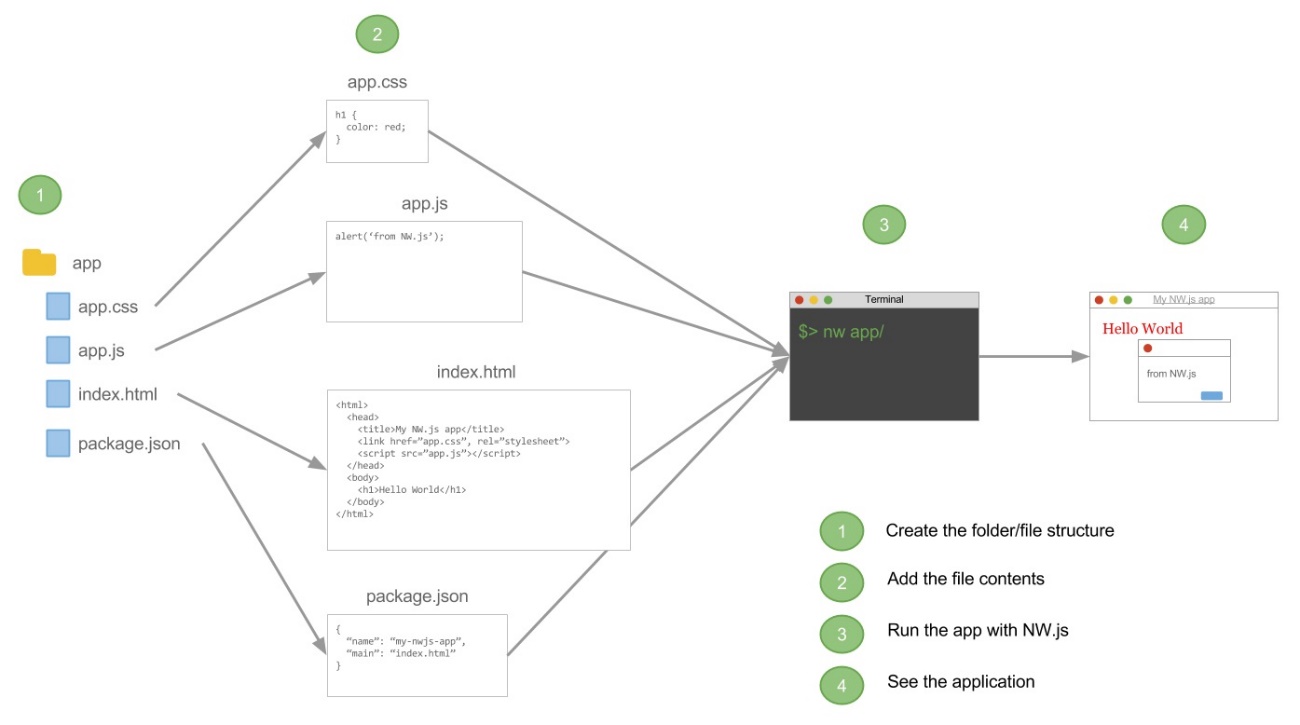
Az Angular egy egyoldalas böngészős kliensalkalmazás fejlesztésére alkalmas TypeScript alapú nyílt forráskódú keretrendszer. A fejlesztés TypeScript, HTML, valamint CSS használatával történik a keretrendszerben. Az előnye, hogy egyszerű böngészős kliensalkalmazást lehet vele létrehozni, amely szükséges a szerveralkalmazás teszteléséhez. A hátránya, hogy a feladatban leírt hálózati megoldásokat nem lehet alkalmazni vele, mivel böngészőből nem lehetséges TCP kapcsolatot nyitni , csak HTTP/HTTPS kapcsolatot, amelyek a TCP-re épülő kapcsolatok, de mégsem a feladatban leírt elvárt lehetséges hálózati megoldások. [5]

**3.2.3. Cordova**

A Cordova egy nyílt forráskódú elsősorban mobil fejlesztésre használt keretrendszer. Szabványos webes technológiák használatával lehetővé teszi a platformok közötti fejlesztést. Az alkalmazások az egyes platformokra célzott burkolókban futnak, és a szabványoknak megfelelő API-összerendelésekre támaszkodva érik el az egyes eszközök képességeit, például az érzékelőket, az adatokat, a hálózati állapotot stb. Előnye, hogy alkalmas böngészős, mobil és desktop alkalmazás készítésére is. [6]

**3.2.4. NW.js**

Az NW.js egy másik nyílt forráskódú keretrendszer webes alapú desktop alkalmazások készítéséhez. Az NW.js egyik alapja a Node.js. A keretrendszer nem csupán egy webalkalmazást nyit nekünk meg egy alkalmazásablakban, hanem magával az operációs rendszerrel is kapcsolatot tud létesíteni és hozzá tud férni a helyi fájlokhoz.



3. ábra, NW.js alkalmazás felépítése, működése [7]

Az alkalmazás felépítése egy egyszerű webalkalmazáséhoz hasonlítanak, amelyeket böngészőben megnyitva ugyanúgy működne mint NW.js-sel. Az egyetlen különbség a package.json fájl, amit a Node.js használ olyan célokra, mint hogy betöltse az alkalmazásokat és könyvtárakat, az NW.js pedig arra használja, hogy itt tároljon konfigurációs adatokat. Ezen fájl nélkül az NW.js alkalmazásunk nem tud betölteni. Az NW.js be tudja tölteni az alkalmazást, ha megadjuk neki a mappa elérési útvonalát. Ebben a mappában megkeresi a package.json fájlt és onnan már tudja, hogy melyik HTML fájlt kell betöltenie és onnantól fut az alkalmazásunk. [7]

**3.2.5. Összehasonlítás, választás**

A leírt feladathoz az Angular keretrendszer nem felel meg az elvárásoknak, mivel nem desktop alkalmazás fejlesztésére szolgál. Az oka amiért mégis megemlítettem, mint lehetséges megoldás az az, hogy szükséges egy kliens alkalmazás fejlesztése is a szerver alkalmazás teszteléséhez. Kliens alkalmazásnak alkalmas lehetett volna, mivel egyszerű böngészős megoldás lenne, de sajnos böngészőből nem lehet TCP socketet nyitni, ami miatt elvetettem ezt a megoldást a kliens alkalmazás számára is.

A Cordova és az NW.js között elég kevés különbséget lehet felfedezni, mind a kettő alkalmas desktop alkalmazások fejlesztésére különböző platformokon. Az egyetlen nagy különbség az, hogy az NW.js a Node.js-re épül. Számomra a döntő érv pont ez volt, mivel a Node.js modulokat biztosít számunkra szerver-oldali alkalmazásfejlesztéshez, amely jelen esetben a célunk.

**3.3. Base64 kódolás, Data URI**

A fájlok és különböző típusú médiafájlok TCP-n való elküldéséhez szükségünk van egy kódolásra és sémára. Ennek a kódolásnak és sémának a Base64 kódolást és a Data URI-t találtam a legmegfelelőbbnek. A Base64 kódolás a leggyakrabban használt kódolási algoritmus bináris adatok kódolásához, például képek, hangfájlok, videók stb. A felépítése, amiből a neve is jön, az alap 64 karakterből áll, amelyek az angol ábécé kis- és nagybetűi, a számok egytől tízig, valamint a plusz- és perjel. A Base64 algoritmus mindegy egyes 3 byte-ot 4 byte-tal helyettesít, ezért a mérete minden esetben 33%-kal nagyobb lesz, ami hátrány nagy fájlok küldése esetén. [8]

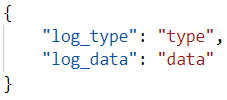
Fájlok küldése esetén nem elegendő magát a kép adat részét elküldenünk a szerverünk számára, szükséges még tudnia a fájl mime type-ját is. A mime type egy szabvány fájlok és dokumentumok formátumának megnevezésére. A mime type egy típusból és egy altípusból épül fel, egy perjellel elválasztva. Mindezért van szükségünk egy sémára, ami tartalmaz egy kódolást és a fájl mime type-ját is. [9] Ilyen sémának, amely a számomra megfelelő jelen esetben a Data URI-t gondoltam, más néven Data URL. A felépítése a Data URL-nek a következő:

data:[<mime type>][;base64],<data>

A Data URL sémában szerepel a fájl mime type-ja, valamint a fájl Base64 kódolt adat része is. A feladat megoldásához épp ezen okok miatt választottam a Data URL sémát, mivel szükségünk lesz a fájlok mime type-jára, valamint fájlok kódolására.

**4. Specifikáció**

Az alkalmazásnak rendelkeznie kell egy kliens és egy szerver oldallal. Az alkalmazás fejlesztéséhez NW.js keretrendszert használok, Node.js modulokat, valamint TCP-t mint hálózati protokollt. A logok megjelenítéséhez szükségünk van egy sémára vagy sémákra, amelyeket fel kell dolgoznunk, ilyen sémának a JSON-t gondoltam alap sémának, ezen belül is a log típusának, valamint a log adatának megadásával az alábbi módon.



4. ábra, JSON log felépítése

A következő alfejezetekben az alkalmazás használati eseteiről, funkcióiról és arról lesz szó, hogy mit is kell tudnia az alkalmazásnak.

**4.1. Használati esetek**

A felhasználónak képesnek kell lennie az alábbiak végrehajtására:

* Logok monitorozása, felügyelete típusuknak megfelelő megjelenítés mellett
* Médiafájlok (kép, videó, hangfájl) megtekintése, lejátszása
* Új kliens megnyitása tesztüzenet küldésére
* A kliens ablakban fájlok kiválasztására és elküldésére az adott hosztról

**4.2. Funkciók**

Az alkalmazásnak egy megadott porton hallgatóznia kell. Amikor ezen a portszámon adat jön, azt maximum 64KB-os csomagokban kapja meg amit, ha nagyobb fájl érkezik, össze kell fűznie az eredeti üzenetté. Az eredeti üzenetet továbbadja feldolgozásra, ahol megnézi milyen formában is kapta az adatot.

Az alkalmazás sorban végig próbálja a kezelő moduljainkat/pluginjainkat amelyek a megfelelő mappánkban találhatóak. Ebbe a mappába utólag is helyezhetőnek kell lennie új kezelő JavaScript fájloknak. A mappa kezelőinek sorban átadjuk a kapott adatot majd, ha siker feldolgoznia visszaadja nekünk a megjelenítésre szánt elemünket. Ha nem sikerült feldolgoznia a kezelőnek, a következő kezelő próbálja meg feldolgozni. Ez addig folytatódik amíg vagy nem sikerül feldolgoznia egy kezelőnknek, vagy végig értünk a kezelőkön a mappánkban. Abban az esetben, ha sikerült feldolgoznia az adatot egy kezelő modulnak, megjelenítjük időbélyeggel ellátva a visszakapott elemet. Ellenkező esetben, ha a kezelő moduljainknak nem sikerült az adat típusának megfelelően feldolgozni, akkor egy alap kezelőt hívunk meg. Ez a kezelő megjeleníti nekünk „<timestamp>: <mime type>: <data>” alakban (hasonlóan a sikeresen feldolgozott üzenetekhez). Ha a kapott csomagnak maga az adat része rövidebb 1024 karakternél akkor kiírjuk az előbb említett alakban, ha több mint 1024 karakter akkor pedig egyszerűen kiírjuk, hogy túl nagy az adatcsomag és nem sikerült kezelni. A kapott adatcsomagokat minden esetben ellátjuk egy időbélyeggel, mivel ez fontos része egy monitorozó alkalmazásnak.

Egy ilyen alkalmazástól elvárható, hogy tárolja, elmentse a logokat, ezért az alkalmazás a megjelenítéssel együtt egy text fájlba el is tárolja a logokat, későbbi visszakeresésekhez vagy egyéb okok miatt. A fájlunk neve mindig az adott napi dátum, a logok pedig hasonlóan néznek ki a fájlunkban mint ahogy megjelenítjük őket az alkalmazás felületén.

Az alkalmazás monitorozó felületén egy menüsorban megtalálható egy „Open new Client” gomb. A gomb megnyomásával tesztelés céljára megnyílik egy új ablak egy kliens felülettel. Ezen a kliens felületen próba logokat lehet küldeni a szervernek. Ezen a felületen található egy fájlválasztó, aminek használatával ki lehet választani a hosztunkról tetszőleges fájlt a szervernek szánt elküldésre. Ezt a fájlt a kliens elküldi a monitorozó alkalmazásnak. Nagyobb, mint 64KB méretű fájl esetén több csomagban küldi el. Amint a fájlt sikerült átküldeni a szerver oldalnak, a kliens oldal zárja a kapcsolatot a szerver oldallal.

**5. Munkafázisok és tapasztalatok**

Ebben a fejezetben végig megyek a fejlesztés munkafázisain, leírom a munkafázisokhoz tartozó tapasztalataimat és a végén a felmerülő problémáimat is összegzem.

**5.1. Adatküldés és fogadás**

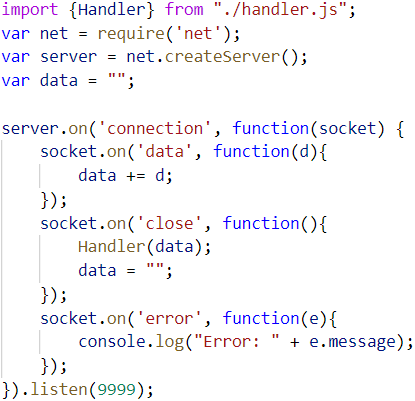
Az első munkafázis egy egyszerű adatküldő kliens alkalmazás és fogadó szerver alkalmazás elkészítése volt. A megvalósításhoz Node.js net modulját használtam, amellyel TCP socketet lehet nyitni és adatot lehet küldeni, illetve fogadni rajta. Egyszerű szöveges vagy JSON típusú log üzenet küldéséhez készítettem egy terminálból futtatható Node.js alkalmazást. A következő ábrán ezen alkalmazás kódja látható.



5. ábra, Egyszerű Node.js kliens alkalmazás

A kódrészletben azt látjuk, hogy egy JSON típusú fájlt és a Node.js net modulját importáltam be a Node.js require funkciójával. Utána beállításnak megadtam egy portszámot, majd ezt a létrehozott client-nek megadtam beállításként és meg is nyitja a kapcsolatot a szerverrel. Utána a JSON típusú fájlomat szöveggé alakítom majd a client connect eseménye bekövetkeztekor lefuttatom a függvényemet, amelyben a write metódussal, a data mint paraméter megadásával elküldi a data-t a szervernek. Végül pedig bontom a kapcsolatot a szerverrel.

Ez az egyszerű Node,js alkalmazás tökéletes számunka szöveges üzenetek vagy JSON típusú fájlok küldésére a szervernek.



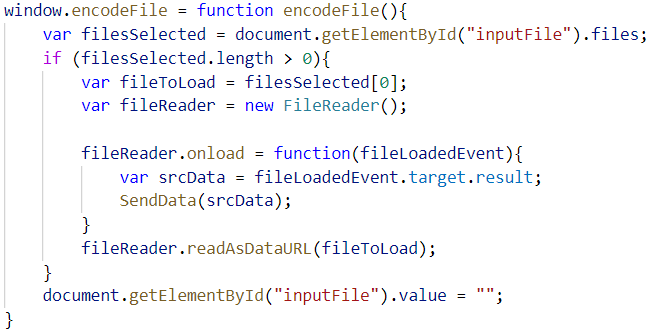
6. ábra, Szerver-oldal kódrészlet

A fenti kódrészleten a szerver oldalt látjuk, ahol szintén importáljuk ugyanazt a Node.js net modult, mint a kliens alkalmazásunknál, valamint a kezelőválasztó függvényünket, amelyről későbbi fejezetben lesz szó. Ezek után létre is tudjuk hozni a szervert majd a különböző eseményeihez a szerverünknek függvényekkel meg tudjuk adni, hogy mit hajtson végre. Az első eseményünk a data esemény, ami annyit jelent, hogy adatot kaptunk valahonnan. Abban az esetben, ha adatot kaptunk szeretnénk, ha azt egy változóba eltárolnánk amíg nincsen kezelve. Mivel TCP kapcsolaton kapjuk az adatainkat ezért maximum 64 KB-os csomagokat kapunk, tehát ha nagyobb adatcsomag érkezik akkor azt több darabban kapjuk meg. Mindezért a data változónkhoz hozzáfűzünk minden egyes d csomagot, amit kapunk mielőtt a kapcsolat megszűnne. A következő eseményünk a close, ahol a kapcsolat bontása esetén fut le az ott megadott kezelőválasztó függvényünk. A kliensünk a kapcsolatot lezárja miután elküldte a számunkra szánt adatcsomagját ezért, a kapcsolat bontásánál meghívjuk a kezelőválasztó függvényünket, amelynek megadjuk paraméterként a kapott adatot, utána a data értékét visszaállítjuk, hogy ne tartalmazzon semmit a következő csomagunk érkezésénél. Az utolsó eseményünk az error esemény, ahol meg tudjuk mondani, hogy mi történjen hiba esetén. Hiba esetén mi most csak az error üzenetet szeretnénk kiírni console-ra. A kódrészlet utolsó sorában a listen metódust használva, a használni kívánt portszám paraméterkénti megadásával tudunk hallgatózni ezen a porton.

A fejezetben bemutattam, ahogy létrehoztam egy egyszerű kliens alkalmazást, amelyet terminálból futtatva tudunk küldeni egy szöveges üzenetet vagy egy JSON típusú fájl. A későbbiekben egy hasonló klienst fogok használni annyi különbséggel, hogy egy függvénytől kapja az adatot paraméternek, amit elküld, nem pedig egy statikusan megadott üzenetet fog elküldeni. A fejezetben szó volt a szerver oldali adatcsomagok fogadásáról is, későbbi fejezetben pedig szó lesz az adatcsomagok kezeléséről és megjelenítéséről is szerver oldalon.

**5.2. Fájlok küldése**

Az alábbi kódrészletben található az alkalmazásunkba épített egyszerű kliens alkalmazás fájlválasztó függvénye. Ezen függvénnyel a kliens ablakunkban kiválasztott fájlt olvassuk be. Az elemünket az id-ja szerint egy változóban eltároljuk. Ez után egy új FileReader-t hozunk létre és megadjuk, hogy Data URL-ként akarjuk beolvasni a fájlt. A load eseménnyel beolvassuk a fájlt és a kapott paraméter Data URL-ét kapjuk meg amit egy változóban eltárolunk majd továbbadjuk a kliens alkalmazásnak elküldésre.

****

7. ábra, Fájlmegnyitása és Data URL továbbadása

Az előző alfejezetben említett egyszerű Node.js kliens alkalmazással, az előbb leírt fájl beolvasással, valamint annak Data URL-jének továbbadásával a küldő függvényünk számára, a kliens oldalt késznek tekintem, mivel tudunk JSON típusú logokat és bármilyen egyéb típusú fájlt küldeni.

**5.3. Adatkezelés, kezelő modulok**

A következő alfejezetekben a modulok kezeléséről és a különböző modulok működéséről lesz szó, valamint arról, hogyan is lehet bővíthető az alkalmazás, milyen feltételeknek kell a moduloknak megfelelnie az alkalmazás egyszerűbb bővíthetőségének érdekében.

**5.3.1. Modulok kezelése**

Az következő kódrészlet a modulok kezelését, valamint az ismeretlen adatokat kezeli úgy, hogy először végig fésüljük a handler-plugins mappánkat JavaScript fájlokat keresve.

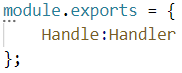
****

8. ábra, Modulok kezelése

A végig fésüléshez a glob modult használjuk, és annak a sync metódusát. A sync metódus annyit jelent, hogy szinkronikusan keressük a fájlokat azért, hogy addig ne menjen tovább a folyamat, amíg nem töltöttük fel a plugins tömbünket. A metódusnak megadjuk a mappánk elérési útvonalát úgy, hogy a relatív elérési útvonalat és a keresni kívánt kiterjesztést abszolút elérési útvonallá alakítjuk. Utána minden egyes fájlt végig nézve a mappánkban a megfelelő kiterjesztésű fájlokat eltároljuk a plugins tömbünkbe a Node.js require funkciójával, a fájlok abszolút elérési útvonalainak megadásával.

A plugins tömbünk feltöltése után egy for ciklussal az összes elemén végig megyünk és az elsőnél, amely hiba nélkül tudja kezelni a kapott adatot, eltároljuk a visszaadott értéket és meghívjuk a megjelenítő függvényünket, amelyről későbbi alfejezetben fogok tárgyalni. A megjelenítő függvény meghívása után a ciklusváltozó növelésével véget is vetünk a ciklusnak, annak érdekében, hogy ha nagyon sok modulunk lenne később akkor ne menjünk végig mindegyiken sok időt elpazarolva és hogy nehogy többször legyen kezelve és többször legyen megjelenítve ugyanaz a log. Abban az esetben, ha elértük a tömbünk utolsó tagját és még nem dolgoztuk fel, meghívjuk az Unknown\_Handler alap kezelőt.

A modulok készítéséhez szükséges néhány feltételnek megfelelnie, hogy az alkalmazás tudja használni. Bemeneti paraméterként csak a nyers adatot kell elkérnie és egy HTML elemet kell visszaadnia, amelyet a megjelenítő függvény fel tud használni. Az alábbi kódrészleten látható hogyan szükséges exportálni a modult és visszaadnia az értéket ahhoz, hogy a Node.js require funkciója importálni tudja.

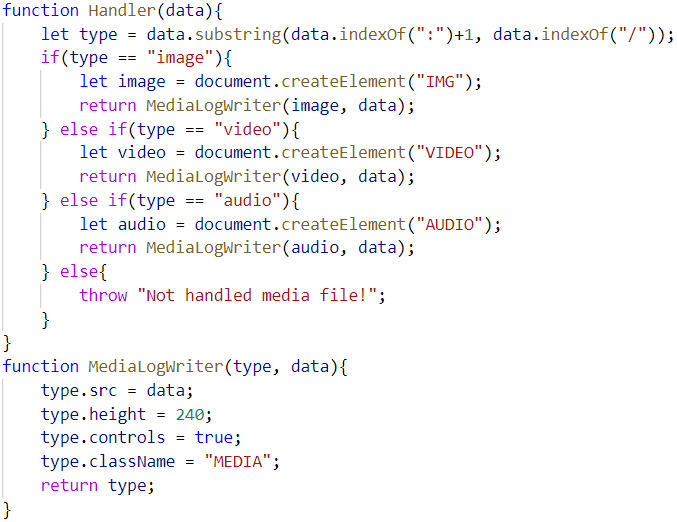
****

9. ábra, Modul exportálása

Másképpen exportálva az alkalmazás nem tudja importálni a modulokat. A modulkezelő függvényünkben minden egyes modulnak a Handle függvényét hívjuk meg, ezért minden modulnak meg kell adnia a Handle exportált értéknél a modul kezelő függvényét, amely jelen esetünkben a Handler függvény.

**5.3.2. Base64 modul**

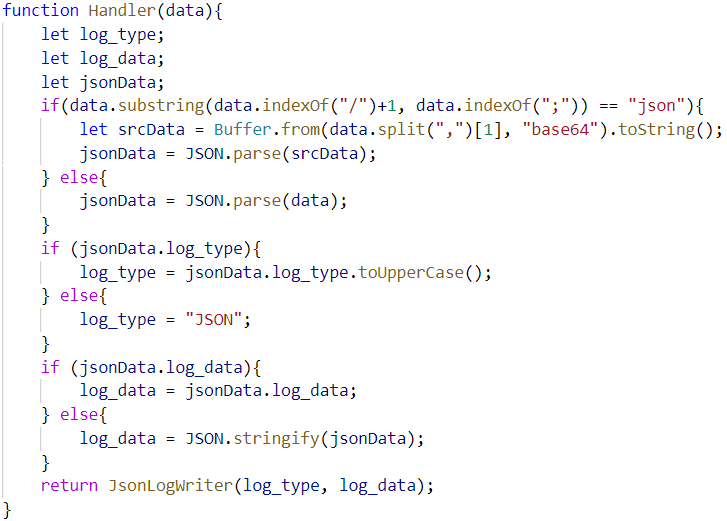
A Base64 médiafájl kezelő modulunk miután megkapta a nyers adatot a modulkezelőtől, a nyers adatból, Data URL-ből kiszedjük a típusát a kapott csomagnak. Ezt a típust megvizsgáljuk, hogy kép, videó vagy esetleg hangfájlt kaptunk-e. Abban az esetben, ha egyiket sem, hibát dobunk, és a modulkezelőnk a következő modullal próbálkozik feldolgozni az adatot. Kép, videó vagy hangfájl esetén létrehozunk egy a típusnak megfelelő HTML elemet, majd ennek az elemnek beállítjuk az attribútumait. Megadjuk neki forrásnak a teljes adatot, amit kaptunk, megadunk egy magasságot, hogy ne foglaljon nagyon sok helyet a megjelenítésnél az esetleges nagy felbontású kép, illetve videó. Megadjuk, hogy lehessen a fájlt irányítani, ez csak kép esetén nem használható, videó és hangfájl esetén megjelennek a megfelelő gombok, valamint az osztályát beállítjuk, a későbbi típus szerinti megfelelő megjelenítés céljából. Mindezek után már csak az előző alfejezetben említett módon szükséges visszaadnunk a HTML elemet és a Base64 kódolású médiafájlok kezelve is vannak. Az alábbi kódrészleten látható pontosan, hogyan is vannak kezelve a Data URL-ként kapott Base64 kódolt médiafájlok.



10. ábra, Médiafájlok kezelése, kódrészlet

**5.3.3. JSON modul**

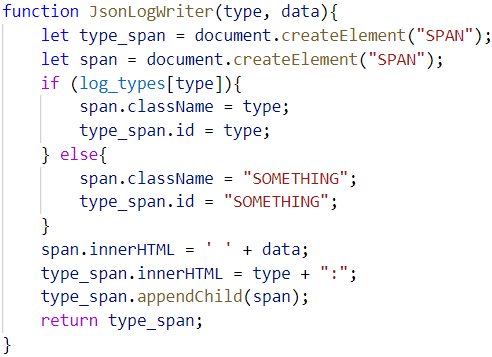
A következő kezelő modulunk megvizsgálja, hogy JSON típusú adatot kaptunk-e. Ezt többféleképpen is megteszi. Megnézi, hogy Data URL-ként jött-e az adat vagy esetleg egyszerű szöveggé alakítva. Ezt meg tudjuk állapítani úgy, hogy a nyers adatban megnézzük, hogy a Data URL felépítése szerint a / és ; között az érkezett adat altípusa „json” akkor annak megfelelően használjuk fel, és az adat részt Base64 dekódoljuk. Ha nem „json” az altípus akkor megpróbáljuk egyszerűen a kapott adatot visszaalakítani JavaScript objektummá. Abban az esetben, ha nem sikerül egyik sem, hiba dobásával visszatérünk a modulkezelőnkhöz és próbálkozunk a következő modullal feldolgozni az adatot. A következő ábrán látható a JSON típusú logokat feldolgozó kódrészlet.



11. ábra, JSON típusú logok kezelése

Miután sikerült rájönnünk, hogy JSON típusú adatot kaptunk, megnézzük, hogy létezik-e log\_type és log\_data értéke az objektumunknak és ha léteznek, azokat eltároljuk. Log\_type érték hiánya esetén „JSON”-t adunk meg megjelenítendő típusnak, log\_data érték hiánya esetén a teljes adatot szöveggé alakítjuk, és majd úgy írjuk ki. Ezek után meghívjuk a következő függvényünket, amely a modulkezelő által visszavárt értéket hozza létre nekünk. A függvényünk összefűzi a log\_type és a log\_data értékeket egy span element értékeként, valamint megadjuk az osztályát ennek az elemnek, amelynek a megjelenítési stílusnál vesszük hasznát.

A következő kódrészlet azt mutatja nekünk, hogy milyen HTML elemet is adunk vissza a modulkezelőnknek. Igényesebb megjelenítés végett szükséges külön id-t adni a type\_span elemünknek. A megjelenítéssel foglalkozó következő fejezetben kifejtem, hogy miért is szükséges mindez.



12. ábra, JSON kezelő modul visszaadott elemének előállítása

**5.3.4. Alap kezelő**

Az alap kezelőnk dolga, hogy minden bejövő adatot feldolgozzon, amit nem sikerült a moduloknak. Ezt nagyon egyszerűen tesszük, nincs szükségünk teljesen ismeretlen adatok bonyolult feldolgozására.

Először megnézzük, hogy Data URL-t kaptunk-e és ha igen akkor a Base64 kódolt adat részét dekódoljuk, valamint a teljes mime type-ot is kiszedjük a Data URL-ből. Ha ezek nem sikerülnek, akkor a kiírandó szövegünk a megkapott nyers adat lesz szövegként. Abban az esetben, ha a kiírandó adat rövidebb ezer karakternél, kiírjuk az adatot, de ha több mint ezer karakter, csak a típust írjuk ki és hogy túl hosszú az adatrész, mindezt azért, hogy ne töltsük meg a teljes alkalmazást egyetlen hosszú üzenettel.

**5.4. Megjelenítés**

A megjelenítését, valamint időbélyeggel ellátását a logoknak és a kezelt egyéb fájloknak a következő kódrészlet mutatja.



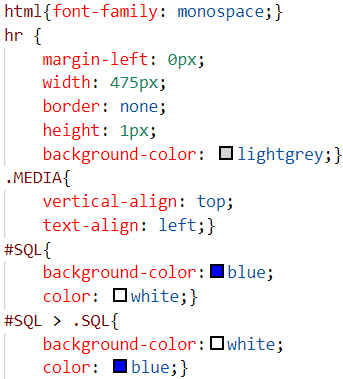
13. ábra, Megjelenítés kódrészlet

A logok megjelenítéséhez szükséges egy időbélyeggel ellátnunk őket. Én az időbélyegnek a hh:mm:ss formátumot választotta, mivel véleményem szerint elegendő másodperc pontosan megjelenítenünk a logokat. Az időbélyeget egy HTML elembe helyezzük és megadjuk az elem osztályát is. A felhasználáshoz megfelelő formátumot a későbbiekben egyszerűen lehet változtatni a kódban.

A megjelenítő függvényünk megkapja a modulok visszaadott HTML elemét és ezt az elemet az időbélyeggel ellátva megjeleníti nekünk, de előtte meghívjuk a LogSaver függvényünket, melyről a következő fejezetben tárgyalok. Minden egyes különböző típusú lognak más-más az osztálya azért, hogy típusuknak megfelelő színnel és akár stílussal lehessen őket megjeleníteni. A JSON típusú logoknál a típusnak külön id-ja van, hogy más módon kezelhessük a visszakapott elemen belül is.

A logok megjelenítéséhez egy monospace betűtípust használunk, hogy igényesebben és szebben nézzenek ki az egymás fölött/alatt megjelenő logok. A különböző típusú logokat más-más színnel jelenítjük meg, valamint egy vékony szürke vonallal elválasztjuk egymástól, a könnyebb olvashatóság végett. A logok típusát a megfelelő színű háttérrel és fehér szöveggel jelenítjük meg, a logok szövegét pedig ellentétesen, fehér háttéren és a szöveget pedig a megfelelő színnel ellátva. A megjelenítéssel foglalkozó függvényben minden beérkező log után legörgetjük az alkalmazás megjelenítő felületét is automatikusan a logok könnyebb nyomon követése végett.

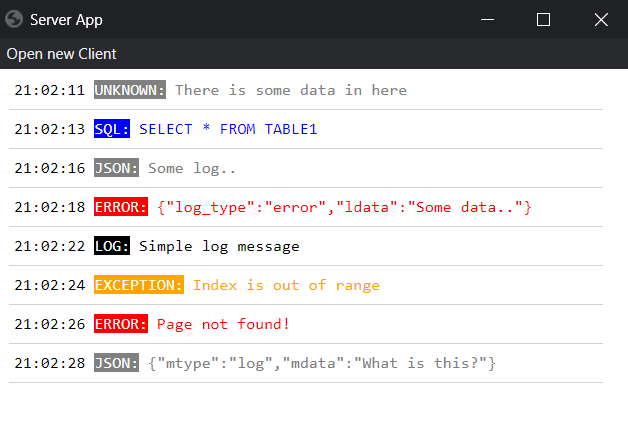
Magát a logok megjelenítési stílusát egy CSS állományban írjuk le a következőképpen.



14. ábra, Megjelenítési beállítások, kódrészlet

Az egész HTML-re megadjuk beállításként a monospace betűtípus-családot, amely annyit takar, hogy egyenlő méretűek horizontálisan a karakterek. A médiafájlok megjelenítésénél szükségünk van a HTML elemünkben bal fentre igazítani az időbélyeget, hogy esztétikusan tudjuk megjeleníteni a képeket, videókat, hangfájlokat. Az ábrán látható az SQL típusú log megjelenítési beállítása. Más típusú fájlok megjelenítése ugyanígy működik, csak a saját megjelenítési színét használva.

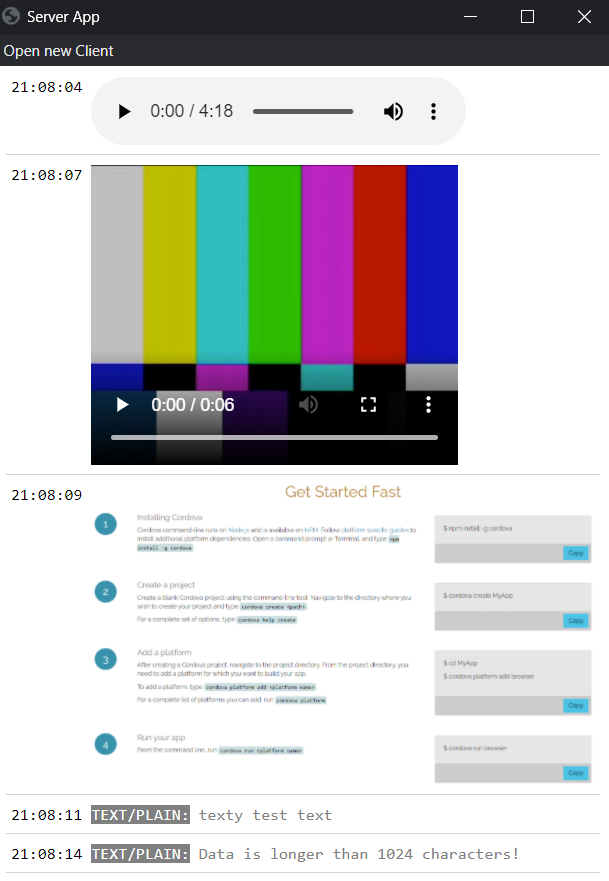
A 15. ábrán látható, hogyan is jelennek meg a különböző típusú kezelt JSON logok, valamint az alkalmazásnak a megjelenése. A menüsorban található az új kliens megnyitására alkalmas gombunk, ahonnan teszt üzeneteket küldhetünk megjelenítésre. Az ábrán látható minden jelenleg kezelt JSON típusú log megjelenítése.



15. ábra, JSON típusú logok megjelenítése

Az első megjelenített logunk egy UNKNOWN log\_type-al rendelkező log volt, ezért szürkén írjuk ki, mint a többi ismeretlen vagy kezeletlen log típust. A második egy SQL parancs volt, amit kéken jelenítünk meg. A következő egy olyan JSON volt, amelynek nem volt log\_type értéke, ezért nem tudjuk kiírni azt, de volt log\_data értéke, ezért azt megfelelően ki tudjuk írni. A következő ERROR üzenetnél pont fordítva van, létezik log\_type értéke, de nincs lod\_data értéke ezért kiírjuk az összes tartalmát a kapott csomagnak. Ezek után találunk három jól kezelt JSON logot (LOG, EXCEPTION, ERROR), majd található egy rossz log\_type és rossz log\_data értékkel rendelkező JSON log.

A 16. ábrán láthatunk egy hangfájlt, egy videót és egy képet is megjelenítve. Utána található egy TXT fájl, ami rövidebb, mint ezer karakter hosszú, ezért kiírtuk a tartalmát. Majd a végén található egy 1024 karakternél hosszabb adattal bíró TXT fájl. Minden kapott adat fel van dolgozva az előbbi módon, megfelelő kezelés nélkül, de megjelenítjük akármilyen adatot is kaptunk a kapott adat hosszától függően.

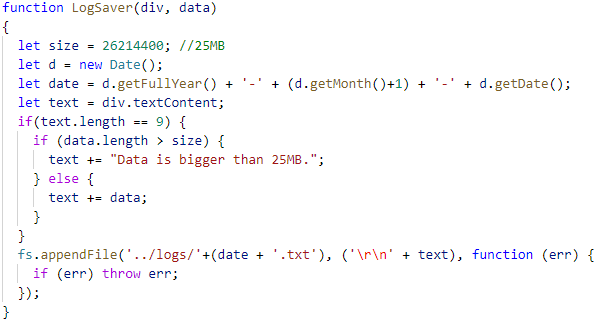


16. ábra, Média fájlok és kezeletlen adatok megjelenítése

**5.5. Logok mentése**

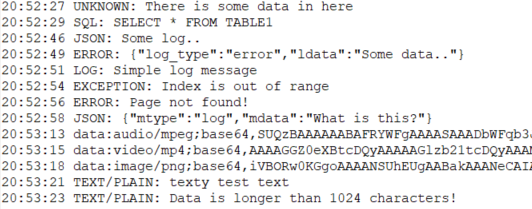
A logokat egy text fájlba mentjük ki, és minden egyes sorában egy log található. A szöveges logok egyszerűen időbélyeggel ellátva vannak megjelenítve hasonlóan, mint az alkalmazásban, leszámítva a különböző stílusokat. A képek, videók, hangfájloknak az időbélyeg és a nyers Data URL szerepel a sorában, abban az esetben, ha kisebbek, mint 25MB. Ezt egy változó változtatásával lehet módosítani. Úgy gondoltam, hogy a nagy méretű fájloknak nincsen keresni valója egy ilyen szöveges fájlban, ezeket később lehetne tárolni saját kiterjesztésével, és a logok között a text fájlban egy ID-val ellátva lehetne hivatkozni a fájlra.

A következő kódrészlet mutatja, hogyan is írjuk text fájlba a logokat.



17. ábra, Logok mentése, kódrészlet

A mentés függvényünk megkapja a megjelenített teljes HTML elemet és a nyers adatot. Létrehozunk egy dátum változót, amit a fájlunk nevének adunk. Minden nap logjait egy külön text fájlban tároljuk, a könnyebb visszakereshetőség és kezelhetőség végett. A megkapott HTML elemünknek a textContent tulajdonságával egy változóban eltároljuk a szöveges adatokat az elemünkből. Ha ez a szöveges adat kilenc karakter hosszú, akkor olyan logról beszélünk, ami egy média fájl lesz, mivel csak az időbélyeg szerepel szövegként a HTML elemben. Ebben az esetben megvizsgáljuk a méretét és ha kisebb, mint 25MB kimentjük a teljes Data URL-t időbélyeggel ellátva. Ha nagyobb akkor a fájlunkba az időbélyeg után írjuk, hogy nagyobb, mint 25MB. A következő ábrán láthatjuk, hogyan is mentjük text fájlba a különböző típusú logokat.



18. ábra, Mentett log napló

**5.6. Tapasztalataim és felmerülő problémák**

A feladatot kihívásnak találtam sok szempontokból, voltak felmerülő problémák, melyeket le kellett győzni. A fejezet ezen felmerülő problémákról szól, valamint tapasztalataimról.

A program fejlesztéséhez szükséges volt egy console-ra, amelyre ki tudok írni tesz üzeneteket, hogy éppen hol akad el a program, vagy megkaptam-e a megfelelő adatot stb. Böngészőből egyszerűen meg lehet nyitni a DevTools nevezetű webfejlesztéshez használt eszközt. Rövid idő után rájöttem, hogyan tudom ezt asztali alkalmazás indításakor is megnyitni, megjeleníteni.

A TCP alapú hálózati kapcsolatot nem volt kihívás felépíteni, és adatot is küldeni TCP kapcsolaton keresztül. Problémát az jelentett, hogy először egy Angular alapú egyszerű webalkalmazást képzeltem el a kliens alkalmazásnak. Sok utána olvasás után beláttam, hogy egyszerűen és szépen nem lehet megvalósítani, amit szeretnék, mert nem lehet böngészőből TCP socketet nyitni, csak valamennyi ráépülő protokollt lehet használni ilyen célokra.

A fájlok átküldésénél kevés problémába ütköztem. A Base64 kódolást hamar alkalmazni tudtam, viszont a Data URL készítése, valamint dekódolása (nem médiafájlok esetén) nagyobb feladatnak tűnt, mint gondoltam. Először Base64 kódolva szerettem volna minden üzenetet elküldeni, de amikor rátaláltam a Data URL sémára, tudtam, hogy a fájlok könnyebb kezeléséhez vele szükséges megoldanom az adatok küldését és később a feldolgozását.

A kezelő modulok írásánál nem akadtak problémáim, viszont sok esetre illik felkészíteni a programot, ha például nem az elvárt módon kapunk adatot azt is lehetőség szerint dolgozzuk fel, minél többféle adatot lefedve a kezeléssel. A legtöbbet időmet a munkafázisok közül a modulok kezelésével töltöttem, hogy az utólag beillesztett modulokat felismerje a programunk, és fel is használja. Sok lehetséges megoldást végig gondoltam és próbáltam mire, megtaláltam azt amelyikkel megvalósítható a feladat. A JavaScript alap importálását nem lehet használni függvényeken belül, ezért a Node.js require funkcióját kellett használnom. A kódban require funkciót egyedül Node.js modulok importálására használtam, ezért új volt számomra saját JavaScript fájlok importálása ezzel a megoldással. Itt olyan problémába ütköztem, hogy a relatív útvonalat nem fogadta el, mint az alap import. Egy idő után rájöttem, hogy abszolút útvonalat használva működhet a kód, valamint a mappa nevében eredetileg alsóvonás volt, melyet szintén nem fogadott el megfelelően a Node.js require metódusa.

A megjelenítés az egyetemen tanultak után nem volt új számomra, egyszerűen meg tudtam valósítani a tanultak alapján. Problémát mégis jelentett a videók megjelenítése. A videók esetén MP4 típusú fájlokat sikerült megjeleníteni és lejátszhatóak is, de hangja nincsen a videóknak. MKV típusú videókat egyáltalán nem tud megjeleníteni az alkalmazás, ezt másik modullal később esetleg meg lehetne valósítani, mint továbbfejlesztési lehetőség.

**7. Használat és továbbfejlesztés**

A következő két alfejezetben szó lesz az alkalmazás használatáról, valamint a lehetséges továbbfejlesztési ötleteimet, gondolataimat is leírom.

**7.1. Alkalmazás használata**

A program használatához szükségünk van az NPM (Node Package Manager) csomagkezelőre. A csomagkezelő használatával tudjuk telepíteni a szükséges, package.json konfigurációs fájlunkban megadott függőségeinket (dependencies) amelyek szükségesek az alkalmazás futásához. Ezek után terminálból „npm run dev” paranccsal tudjuk futtatni az alkalmazást. Az „npm run prod” parancs pedig a konfigurációs fájlokban megadott platformokra fordítja a programot.

A program használata indítás után egyszerű. A monitorozó felületünkön egyetlen gomb található az ablak menüsorában, az új kliens megnyitása gomb. Ezen gomb segítségével egy új kliens ablakot nyithatunk meg, ahol fájlokat választhatunk ki küldésre. Amint kiválasztottuk a fájlt, el is küldi a kliens alkalmazás és megjeleníti a szerver alkalmazás a főablak felületén.

A lementett logokat az alkalmazásunk melletti mappában találhatjuk meg, és a különböző új feldolgozó modulokat is egy az alkalmazás melletti mappába tudjuk elhelyezni.

**7.2. Továbbfejlesztési lehetőségek**

Az alkalmazást sokféleképpen tovább lehet fejleszteni, de leginkább a felhasználástól függ, hogyan is lehetne. A legfontosabb továbbfejlesztési lehetőség szerintem az új modulok készítése, hogy minél több fajta és típusú fájlt és adatot le tudjunk fedni a megfelelő megjelenítéshez. Rengeteg más sémával is kaphatunk adatot, nem csak Data URL-lel és nem csak JSON típusú fájlok lehetnek, amellyel a logokat megkapjuk. Én ezeket választottam a feladatom elkészítéséhez, de ezek más számára lehetnek egészen mások is. Ezt meg lehetne úgy is oldani, hogy a modulokat fel lehet tölteni az alkalmazásunkba egy másik ablakon keresztül, valamint ezeket lehetne sorrendbe helyezni, ki- és bekapcsolni, hogy ha tudjuk milyen adatokat várunk, akkor ne teljen az idő azzal, hogy sok modult végig próbálunk mire a megfelelőhöz érünk.

Az alkalmazást személyre szabhatóvá lehetne tenni, hogy mindenki a neki megfelelő hátteret tudja mondjuk kiválasztani, megfelelő színeket a különböző típusú logok megjelenítéséhez. Szükség esetén szűrésre is lehetőség lenne, hogyha csak bizonyos típusú logokat szeretnénk monitorozni, például csak a hibaüzenetekre vagyunk kíváncsiak éppen, akkor a többi logot nem jelenítjük meg amíg vissza nem kapcsoljuk őket. Az alkalmazás indításakor meg lehetne adni azt is, milyen porton kívánunk hallgatózni és jelszóhoz is lehetne kötni az alkalmazás használatát, idegenek hozzáférésének ellehetetlenítése végett.

Mint említettem a logok mentése fejezetemben, tovább lehetne fejleszteni az alkalmazást olyan módon, hogy a text fájlokban a nagy méretű médiafájloknak nem írjuk ki a teljes Data URL részét, hanem lementhetnénk őket egy külön mappába, amire hivatkozni lehetne a text fájlunkban. Ez hasznos lenne olyan szempontból is, hogy ha olyan fájlt kapunk, amelyet nem tudtunk kezelni, azt később valamilyen módon még meg lehetne nyitni. A logokat akár lehetne felhőben is tárolni, hogy bárhonnan megtekinthetőek legyenek, és ne a monitorozó alkalmazás és számítógép előtt kelljen ülni. Szintén meg lehetne valósítani, hogy ezeket a fájlokat megnyissuk az alkalmazással, és megjelenítse megfelelően nekünk újból.

**8. Összegzés**

A szakdolgozatban ismertettem a megoldandó problémát, amely fontos a feladat teljes megértéséhez. A feladat ismertetése után fontos ismertetni a lehetséges megoldási módszereket, hogy részletesebb képet kapjunk a megvalósításhoz. A szakdolgozat ezért ismerteti az előnyeit és a hátrányait a lehetséges megoldási módszereknek. A megoldási módszerek kiválasztásának indoklása után ismertettem a feladat részletes specifikácóját a kiválasztott módszerekkel, valamint az alkalmazás funkcióit. Ezek után már lehetséges az alkalmazás elkészítési folyamatának leírása. Az alkalmazás munkafázisait kódrészletekkel szemléltettem, a könnyebb megértés végett, valamint egy ábrán szemléltettem az alkalmazás megjelenését a jelenleg lehetséges összes különböző típusú adattal. Az alkalmazás működésének megértése után leírtam felmerülő problémáimat és tapasztalataimat. Végezetül a szakdolgozat leírja az alkalmazás használatát és a lehetséges továbbfejlesztési lehetőségeket.

**Ábrajegyzék**

1. ábra, TCP-összeköttetés létesítése, háromutas kézfogás

2. ábra, TCP és Unix Domain Socket sávszélességek összehasonlítása

3. ábra, NW.js alkalmazás felépítése, működése

4. ábra, JSON log felépítése

5. ábra, Egyszerű Node.js kliens alkalmazás

6. ábra, Szerver-oldal kódrészlet

7. ábra, Fájlmegnyitása és Data URL továbbadása

8. ábra, Modulok kezelése

9. ábra, Modul exportálása

10. ábra, Médiafájlok kezelése, kódrészlet

11. ábra, JSON típusú logok kezelése

12. ábra, JSON kezelő modul visszaadott elemének előállítása

13. ábra, Megjelenítés kódrészlet

14. ábra, Megjelenítési beállítások, kódrészlet

15. ábra, JSON típusú logok megjelenítése

16. ábra, Média fájlok és kezeletlen adatok megjelenítése

17. ábra, Logok mentése, kódrészlet

18. ábra, Mentett log napló

**Irodalomjegyzék**

[1] <https://lists.freebsd.org/pipermail/freebsd-performance/2005-February/001143.html>

[[2] https://regi.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0103\_panem\_szamhal/ch06s05.html](%5b2%5d%20https://regi.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0103_panem_szamhal/ch06s05.html) (ábra is)

[3] <https://www.researchgate.net/figure/Performance-Comparison-of-TCP-vs-Unix-Domain-Sockets-as-a-Function-of-Message-Size_fig3_221461399>

[4] <https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm>

[5] <https://angular.io/guide/architecture>

[6] <https://cordova.apache.org/docs/en/latest/guide/overview/index.html>

[7] https://dzone.com/articles/what-is-nwjs

[8] <https://base64.guru/learn/what-is-base64>

[9] https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics\_of\_HTTP/MIME\_types

<https://nodejs.org/api/buffer.html>

https://nodejs.org/api/net.html

# Ábrák

középre rendezve, feliratozva és számozva. Az ábrák számozása a teljes dokumentumban folyamatos, nem kell a fejezetszámozást is belevenni! Figyelni kell az ábrák minőségére, illetve olvashatóságára. Olvashatatlan, kivehetetlen, illetve túlzottan nagy ábrák a szakdolgozatban nem megengedhetőek. Nagyméretű ábrákat mellékletbe hajtogathatnak be. Forráshivatkozás sorszámozással.

# Fejezet- és alfejezet címek (formai ajánlás)

# minden cím előtt és után egy-egy plusz sortávolság, cím utáni első sor kezdése sor elején (nincs tabulálás).

# Fejlécek

A fejlécek tartalma a diplomadolgozat címe, vagy fejezetenként a fejezet címek.

# A tartalomjegyzékről

A jó tartalomjegyzék tükrözi a dolgozat logikai felépítését. Fejezetekből, alfejezetekből és további egységekből áll. Ezek oldalszámát pontosan jelölni kell, a következők szerint:

1. …………………………………………………………………….. x o.

1.1. ……………………………………………………………………y o.

1.1.2

1.2.

1.2.1.

1.2.2.

2. ……………………………………………………………………..z o.

# Az idézésről

*Tartalmi idézés*

Amikor egy szerzőt nem szó szerint, hanem csak összefoglalva idézünk, akkor is meg kell adni a forrást. Ez történhet a szövegen belül név, évszám - előadásra hivatkozás esetén annak helye ill. témája - megjelölésével. Másik lehetőség, hogy **a szövegben jegyzetszámot alkalmazunk és a dolgozat végén az irodalomjegyzék fejezetben adjuk meg a nevet, címet, de ekkor nem kell megadni az oldalszámot.**

***6. Cikk az Internetről: pl.***  
            MOLNÁR L.: Információs vagy tudás társadalom? *Néhány gondolat a tudásról és az információról* In: INCO 2000/01 számából  
[http://www.inco.hu/inco3/tudas/cikk1h.htm](javascript:if(confirm('http://www.inco.hu/inco3/tudas/cikk1h.htm%20%20\n\nThis%20file%20was%20not%20retrieved%20by%20Teleport%20Pro,%20because%20it%20is%20addressed%20on%20a%20domain%20or%20path%20outside%20the%20boundaries%20set%20for%20its%20Starting%20Address.%20%20\n\nDo%20you%20want%20to%20open%20it%20from%20the%20server?'))window.location='http://www.inco.hu/inco3/tudas/cikk1h.htm') /(látogatva:dátum)