

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Hanicz Tamás

Fájlmegosztó keretrendszer fejlesztése RASPBERRY PI eszközre

Konzulens

Dr. Asztalos Márk

BUDAPEST, 2017

Tartalomjegyzék

[Összefoglaló 5](#_Toc499114651)

[Abstract 6](#_Toc499114652)

[1 Bevezetés 7](#_Toc499114653)

[1.1 Felhasználás 8](#_Toc499114654)

[2 Technológiák 9](#_Toc499114655)

[2.1 Frontend 9](#_Toc499114656)

[2.1.1 Angular 9](#_Toc499114657)

[2.1.2 TypeScript 9](#_Toc499114658)

[2.1.3 Bootstrap 9](#_Toc499114659)

[2.2 Backend 9](#_Toc499114660)

[2.2.1 Könytárak 10](#_Toc499114661)

[2.2.2 Dropbox 10](#_Toc499114662)

[2.3 Raspberry PI 10](#_Toc499114663)

[2.3.1 Raspbian 11](#_Toc499114664)

[3 Specifikáció 12](#_Toc499114665)

[3.1 Use Case 13](#_Toc499114666)

[3.1.1 Felhasználót érintő műveletek 14](#_Toc499114667)

[4 Tervezés és fejlesztés 15](#_Toc499114668)

[4.1 Architektúra 15](#_Toc499114669)

[4.2 Fejlesztőkörnyezet bemutatása 15](#_Toc499114670)

[4.3 Telepítő script 16](#_Toc499114671)

[4.4 Adatbázis 16](#_Toc499114672)

[4.4.1 User 17](#_Toc499114673)

[4.4.2 File 18](#_Toc499114674)

[4.4.3 Folder 18](#_Toc499114675)

[4.4.4 File share 19](#_Toc499114676)

[4.4.5 Role 19](#_Toc499114677)

[4.4.6 Credential store 19](#_Toc499114678)

[4.4.7 Log 20](#_Toc499114679)

[4.5 Végpontok 20](#_Toc499114680)

[4.5.1 Autentikáció 21](#_Toc499114681)

[4.5.2 Adatok ellenőrzése 21](#_Toc499114682)

[4.5.3 UsersAPI 22](#_Toc499114683)

[4.5.4 RolesAPI 22](#_Toc499114684)

[4.5.5 LogsAPI 22](#_Toc499114685)

[4.5.6 NotesAPI 23](#_Toc499114686)

[4.5.7 DropboxAPI 24](#_Toc499114687)

[4.5.8 FilesharesAPI 24](#_Toc499114688)

[4.5.9 FilesAPI 25](#_Toc499114689)

[4.6 Modell 27](#_Toc499114690)

[4.6.1 DropboxModel 27](#_Toc499114691)

[4.6.2 CredentialstoreModel 29](#_Toc499114692)

[4.6.3 UsersModel 29](#_Toc499114693)

[4.6.4 RolesModel 30](#_Toc499114694)

[4.6.5 NotesModel 30](#_Toc499114695)

[4.6.6 LogsModel 30](#_Toc499114696)

[4.6.7 FilesModel 31](#_Toc499114697)

[4.6.8 FilesharesModel 33](#_Toc499114698)

[4.6.9 TokensModel 34](#_Toc499114699)

[4.6.10 Felhasználói limitek 34](#_Toc499114700)

[4.6.11 Logolás 35](#_Toc499114701)

[4.7 UI 36](#_Toc499114702)

[4.7.1 Művelet sikeressége 36](#_Toc499114703)

[4.7.2 Bejelentkezés 36](#_Toc499114704)

[4.7.3 Regisztráció 36](#_Toc499114705)

[4.7.4 Files menü 37](#_Toc499114706)

[4.7.5 Notes menü 42](#_Toc499114707)

[4.7.6 User menü 43](#_Toc499114708)

[5 Mérések 46](#_Toc499114709)

[6 Összefoglalás 49](#_Toc499114710)

[6.1 Továbbfejlesztés 49](#_Toc499114711)

[7 Irodalomjegyzék 52](#_Toc499114712)

Hallgatói nyilatkozat

Alulírott **Hanicz Tamás**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2017. 04. 29

...…………………………………………….

Hanicz Tamás

Összefoglaló

Abstract

# Bevezetés

A feladat egy olyan alkalmazás elkészítése, mely a felhasználók által feltöltött fájlokat tárolja és lehetőséget biztosít azok megosztására. A felhasználók bármikor letölthetik, törölhetik és felülírhatják az általuk feltöltött fájlokat. A fájlokhoz a tulajdonosuk engedélye nélkül senki sem férhet hozzá.

A felhasználók a böngészőjük segítségével érhetik el és használhatják az alkalmazás valamennyi szolgáltatását. Tudnak új regisztrációt csinálni vagy korábban létrehozott fiókjukba belépni. Belépés után a felhasználónak lehetősége nyílik az általa korábban feltöltött fájlokat letölteni, törölni vagy teljesen újakat feltölteni. Továbbá lehet feljegyzéseket, szövegeket elmenteni az oldalon keresztül egy beépített jegyzettömb segítségével. A fájlok fájlrendszerbe rendezhetők tetszőleges módon és keresni is lehet közöttük. A felhasználó saját tevékenységeiről lát logbejegyzéseket.

Egy adott fájl megosztásáról a tulajdonosa rendelkezhet. Tetszés szerint megoszthatja bárkivel egy publikus linkkel, vagy adott felhasználókkal email segítségével. Valamennyi fájl megosztásához többféle jogosultsági szint rendelhető. Lehetőség van a fájlok „verziókövetett” tárolására is, ami azt jelenti, hogy egy fájl korábbi verziói is elérhetőek, ezt a rendszer automatikusan elvégzi minden fájl esetén.

A cél az volt, hogy az alkalmazás képes legyen egy másik, nagyobb fájlmegosztóval szinkronizáltan működni, ami azt jelenti, hogy bizonyos fájlok a felhasználó kérésére feltöltődnek oda is vagy onnan letöltésre kerülnek. Ez azért jó, mert a felhasználó számára fontos fájlok több helyen is elérhetőek lesznek így csökkentve a veszélyét annak, hogy elveszik a fájl, továbbá a két rendszer közti összekapcsolás kényelmet ad a felhasználónak.

Az alkalmazás egy Raspberry PI eszközön fut, úgy hogy a kérések kiszolgálásáért felelős logika Python-ban van megírva míg az adatok megjelenítéséért felelős kódrészlethez Angulart használok fel.

## Felhasználás

Felmerülhet a kérdés, hogy miért pont Raspberry PI-ra készült a szoftver, és hogy pontosan milyen célt is szolgál. Az ötlet az volt, hogy készítsek egy olyan szoftvert, melynek segítségével személyes fájljaim, feljegyzéseim bárhonnan elérhetőek legyenek anélkül, hogy azokat feltölteném egy fájlmegosztóra. Mindenképpen olyan hardvert kerestem, ami bár kis teljesítményű, de egész nap hozzáfér a hálózathoz, képes folytonosan üzemelni és nem igényel sok erőforrást ehhez. Mivel valamennyi felsorolt kritériumnak megfelel, választásom a Raspberry PI eszközre esett. A szoftver is olyan igényekkel készült, hogy nem kell sok párhuzamos felhasználót egyidejűleg kiszolgálnia, hiszen csak kis számú felhasználója lesz. Az előnye ennek az, hogy a hálózatra kötve az eszköz viszonylag gyors működés mellett nem vesz el sokat a rendelkezésre álló sávszélességből és természetesen fizikailag sem igényel nagy helyet és csak kevés karbantartást. A hátránya az, hogy egy nagy fájlmegosztó protállal nem tudja felvenni a versenyt és limitek vannak a rendszerben.

# Technológiák

Ebben a fejezetben a rendszer elkészítéséhez felhasznált technológiák kerülnek bemutatásra, valamint, hogy az egyes könyvtárak, kiegészítők milyen pluszt adtak hozzá a szoftverhez.

## Frontend

A frontend Angular, TypeScript és Bootstrap felhasználásával készült.

### Angular

### TypeScript

### Bootstrap

A Bootstrap egy eszköz, amely segítségével gyorsan lehet HTML, CSS és JS fejlesztéseket végezni. Nagyban megkönnyíti a munkát és minimális kóddal lehet szép és tartalmas felhasználói felületet készíteni. Elég csak a fő komponens <head> tag-jében megadni a stylesheet és script elérését és ezek segítségével az oldal betöltésekor ezeket is letölti.

## Backend

Az alkalmazás backend-je Python programozási nyelv felhasználásával készült,amelynek az alapja egy REST API. A REST szolgáltatás azt jelenti, hogy a rendszerben definiált erőforrásokhoz elérési pontokon keresztül lehet hozzáférni, azokat módosítani vagy törölni. A kommunikáció JSON objektumok segítségével történik, ezek reprezentálják az egyes erőforrásokat.

Flask: A backend alapja tehát egy REST API, mely a Flask [1] felhasználásval készült el. A Flask egy Python alapú webes keretrendszer. Könyvtárakat, technológiákat nyújt ahhoz, hogy minél egyszerűbben és hatékonyabban lehessen webes alkalmazásokat elkészíteni. Önmagában nem nyújt adatbázis absztrakciós réteget, így azt a fejlesztőnek kell megvalósítania, vagy felhasználnia valamilyen kiegészítőt.

SQLite: Az alkalmazás működéséhez elengedhetetlen valamilyen adatbázis használata a felhasználói adatok, logok és a felhasználókhoz tartozó fájlok elérésének tárolásához. A választásom az SQLite-ra esett annak gyorsasága és egyszerűsége miatt.

### Könytárak

Passlib: A Passlib egy jelszó „hashelő” könyvtár Pythonhoz. Ennek segítségével „hashelem” a jelszavakat és tárolom el a kapott értéket az adatbázisban. A könyvtárat a belépés és regisztráció funkcionalitásoknál használtam fel, ezek a folyamatok a későbbiekben kerülnek bemutatásra.

SQLAlchemy: Az SQLAlchemy egy Python SQL eszköz, mely ORM tulajdonsággal rendelkezik, nagy segítséget nyújtva ezzel a fejlesztéshez.

Pyjwt: A Pyjwt egy könyvtár, ami kódol és dekódol JSON web tokeneket. A könyvtár az autentikáció során kerül felhasználásra. Belépéskor a kliens kap egy kódolt token-t, melyet minden kérésnél el kell küldenie a szerver felé.

### Dropbox

A Dropbox lett az általam kiválasztott fájlmegosztó portál, amivel összekötöm az alkalmazást, alapvető szolgáltatásokkal rendelkezik, könnyű kezelni és jól érthető API-val rendelkezik. Az oldal ingyenes verzió esetén 2GB helyet biztosít a felhasználó számára. Regisztrálás után lehet új alkalmazásokat létrehozni Dropbox-on belül, itt meg lehet adni, hogy milyen hozzáférést kérjen a felhasználóktól engedélyezéskor. Az alkalmazást 500 db felhasználóhoz lehet hozzárendelni amíg fejlesztés állapotban van a szoftver, több beállítást nem igényel.

## Raspberry PI

Az alkalmazás egy Raspberry Pi 3 B modellen fut. Mivel az erőforrásai végesek, olyan megvalósításra kell törekedni, mely számításba veszi az alábbiakban felsorolt paramétereket.

Az eszköz legfontosabb fizikai paraméterei:

* Quad Core 1.2GHz BCM2837 CPU - 64bit
* 1GB SDRAM
* 16GB-os C10-es microSD kártya
* Raspbian OS

### Raspbian

A Raspberry PI eszközön Raspbian operációs rendszer fut, ami Debian alapú és az eszközre lett optimalizálva [2]. Az OS több előtelepített szoftverrel is rendelkezik és tartozik hozzá grafikus UI. Bár az operációs rendszeren van Python, azt frissíteni kell, mivel csak 3.6-os verzióval fut az alkalmazás helyesen. Továbbá fel kell telepíteni az npm package managert is, hogy a frontendhez tartozó komponensek is felkerüljenek a rendszerre.

# Specifikáció

A feladat egy olyan alkalmazás elkészítése, mely a felhasználók által feltöltött fájlokat tárolja és lehetőséget biztosít azok megosztására. A felhasználók bármikor letölthetik, törölhetik és felülírhatják az általuk feltöltött fájlokat. A fájlokhoz a tulajdonosuk engedélye nélkül senki sem férhet hozzá.

A felhasználók a böngészőjük segítségével érhetik el az alkalmazást. Először egy login oldal köszönti őket, ahol a korábban beregisztrált adataikkal tudnak belépni. A még nem regisztrált felhasználóknak itt van lehetőségük regisztrálni. Regisztráció után a megadott e-mail címre érkezik egy megerősítő e-mail, benne egy aktiváló linkkel. A felhasználó accountja csak ezen link meglátogatása után használható. Bejelentkezés során a jelszót háromszor lehet elrontani, utána a rendszer kizárja a felhasználót és csak e-mail-en keresztül lehet jelszót változtatni.

Belépés után a felhasználónak lehetősége nyílik az általa korábban feltöltött fájlokat letölteni, azokat felülírni vagy esetleg újakat feltölteni. Továbbá lehet feljegyzéseket, szövegeket elmenteni az oldalon keresztül egy beépített jegyzettömb segítségével. A fájlok fájlrendszerbe rendezhetők tetszőleges módon. Törlés után 14 napig minden fájl visszaállítható, utána azonban már nincs rá lehetőség. A felhasználók limitált tárolási területtel rendelkeznek. Ez jelenleg 1 GB, ami a későbbiekben, a rendszer bővítését követően nőni fog. Az oldalon elérhető egy logfile, amiben az elmúlt napok történéseit lehet megtekinteni. Minden felhasználó csak a hozzá tartozó fájlokról lát logbejegyzéseket. A fájlok között keresni is lehet.

Egy adott fájl megosztásáról a tulajdonosa rendelkezhet. Tetszés szerint megoszthatja bárkivel egy publikus link, vagy adott felhasználókkal email segítségével. Valamennyi fájl megosztásához többféle jogosultsági szint rendelhető:

* Read
* Write
* Delete (magában foglalja az előző kettőt is)

Lehetőség van a fájlok „verziókövetett” tárolására is, ami azt jelenti, hogy egy fájl korábbi verziói is elérhetőek, ezt a funkciót azonban külön be kell kapcsolni fájlonként.

A rendszerhez egy felhasználó típus tartozik, ami az egyszerű felhasználót jelöli. Admin jogosultságra nincs szükség, mivel más adataihoz alapvetően nem lehet hozzáférni.

Az alkalmazás képes egy másik, nagyobb fájlmegosztóval szinkronizáltan működni, ami azt jelenti, hogy bizonyos fájlok a felhasználó kérésére feltöltődnek oda is vagy onnan letöltésre kerülnek.

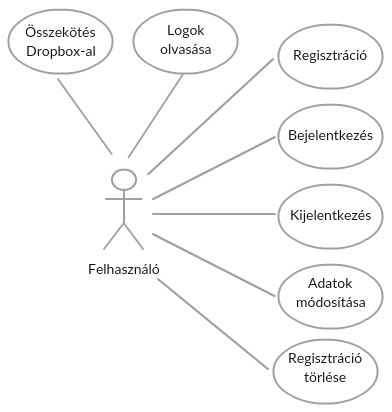
Érdekes kérdés, hogy hogyan lehet leképezni a felhasználó által elképzelt mappa rendszert. Az egyik megoldás az, hogy a felhasználónak van egy fő mappája, amibe bekerül az összes almappa, de nem hierarchia rendszerben, hanem egyszintű lenne, és az adatbázisban tárolt adatok alapján lehetne megállapítani a pontos hierachiát úgy, hogy ebből a felhasználó nem venne észre semmit. Ennek az az előnye, hogy gyorsabb az egyes mappák megtalálása és jobban átlátható, viszont amikor fizikai műveletekre kerül sor, mint például a törlés vagy áthelyezés, akkor nehezebben megvalósítható. A másik megoldás az, hogy pontosan leképezem a hierarchiát, így nehezebbé és hosszabbá téve a mappák megtalálását, de ha egy olyat kell törölnöm, melynek sok gyereke van, akkor az fizikailag egy művelettel megoldható. Végül a második megoldás mellett döntöttem, mert később, ha a rendszert szeretném továbbfejleszteni, akkor ebből még előnyöm származhat.

Ahogy az a technológiáknál bemutattam, a Dropbox a kiválasztott portál, amivel szinkronizáltan működik az alkalmazás. Jól dokumentált és egyszerű API-val rendelkezik, ezért esett rá a választásom [3].

## Use Case

A fejezetben a korábban leírt funkciók tekinthetőek meg Use Case diagramok formájában, melyek segítenek összefoglalni és vizualizálni. A diagrammokat szétbontottam olyan tevékenységekre, melyek vagy csak a felhasználót érintik vagy valamilyen fájlon és note-on végzett műveletek.

### Felhasználót érintő műveletek



# Tervezés és fejlesztés

A fejezetben bemutatom az alkalmazás felépítését és a megvalósítás módszereit, továbbá a kész rendszer felhasználhatóságát képekkel illusztrálva.

## Architektúra

A rendszer több rétegből épül fel,amely fejlesztés szempontjából jól elkülöníthető két fő részre, frontend és backend. A felhasználó csak a felhasználói felületet látja működés során és nem tudja, hogy a háttérben pontosan milyen szolháltatások futnak és az egyes kérések pontosan, hogyan is vannak kiszolgálva.

Ahogy azt korábban említettem a megjelenítésért felelős kódrész Angular és TypeScript felhasználásával készült. A kód komponensekre van felosztva az átláthatóság és újrafelhasználhastóság miatt. Minden komponens egy .ts fájlból, ami a TypeScript kódot tartamazza, egy .html fájlból ami, a HTML kódot tartalmazza és egy .css fájlból épül fel, ami lényegében egy sablon. Minden komponensenek van egy külön mappája amibe csak a hozzátartozó fájlok vannak. A komponenseken kívül van egy entities mappa is, ahova azok az osztályok kerülnek, amelyek gyakran használt objektumokat definiálnak. Továbbá egy services mappa ahova a backend-el kommunikáló kódrészek kerültek.

A backend felépítése is jól felosztható rétegekre, legalul található az adatbázis ami az adatok tárolására szolgál

## Fejlesztőkörnyezet bemutatása

Az alkalmazást a saját gépemen készítettem és teszteltem majd töltöttem át a Raspberry PI eszközre további tesztelés céljából, a szoftver szempontjából csak néhány beállításbeli különbség van, hogy éppen hol fut.

A projekt jól elkülöníthető két részre fejlesztés szempontjából, az egyik a backend míg a másik a frontend. A backend fejlesztéséhez a Pycharm nevű fejlesztőeszközt használtam, amely minden igényt kielégítve nyújt támogatást Pythonban írt projektek fejlesztésére. Könnyen kezelhető benne a projekt, a készített alkalmazást egy gomb nyomással el lehet indítani és indítás után láthatom a kimenetet, tehát nem igényel semmilyen parancssori beavatkozást. Az elkészült REST API-t pedig Postman segítségével teszteltem, hogy megfelelően reagál-e a kapott bementekre. Az SQLAlchemy egyik tulajdonsága, hogy fejlesztés során beállítható, hogy a végrehajtott lekérdezéseket kiírja a kimenetre, ezzel is gyorsítva a fejlesztés menetét, hiszen így könnyebb leellenőrizni, hogy az elkészített utasítások és lekérdezések valóban helyesek-e.

## Telepítő script

A cél egy frissen telepített Raspbian operációs rendszer felkonfigurálása úgy, hogy az alkalmazás probléma nélkül induljon. Ennek eléréshez egy Python-ban megírt scriptet csináltam ami az alábbi parancsokat fogja végrehajtani.Az első feladat frissíteni az operációs rendszerre telepített csomagokat, majd feltelepíteni az npm-et. Mivel a backend futásához Python3.6-ra van szükség, ezért a script azt is letölti majd telepíti. Legvégül pedig létrehozza az SQLite adatbázis fájlt és létrehozza benne a Role és Crendetial\_Store táblákat, amelyekbe beszúr pár rekordot, hogy az alkalmazás hiba nélkül tudjon indulni.

## Adatbázis

A rendszer helyes működéséhez elengedhetetlen az adatok perzisztálása. A rendzser adatbázis struktúrája, annak kialakítása, továbbá az egyes táblák szerepe és a közöttük lévő kapcsolatok kerülnek bemutatásra az alábbi fejezetben. Ahogy azt a technológiák fejezetben leírtam, a program SQLite adatbázist használ. Az SQLAlchemy könyvtár felhasználásával elég csak megadnom az adatbázis típusát illetve annak elérését, így később, ha szeretnék más adatbázisra váltani, akkor elég csak az alábbi kódrészletet átírnom. Más módosítást nem igényel.

engine = create\_engine(**'sqlite:///test.db'**, echo=**True**)

A program induláskor a definiált osztályok alapján megnézi, hogy a hozzájuk asszociált táblák valóban léteznek-e az adatbázisban. Amennyiben van olyan tábla amelyik nem létezik, akkor azt létrehozza.



### User

A felhasználók adatait tárolja, hogy a rendszer azonosítani tudja őket felhasználónév – jelszó páros segítségével. Emellett segít megkülönböztetni a még nem aktivált felhasználókat az aktiváltaktól és felfüggesztettektől, továbbá tárolja a rontott jelszavak számát. A jelszó „hashelve” kerül eltárolásra, növelve ezzel a rendszer biztonságát.

Atttribútumok:

|  |  |
| --- | --- |
| id | felhasználó egyedi azonosítója |
| name | választott név |
| email | e-mail cím, amire az aktiváló e-mail megy |
| password\_hash | jelszó „hashelve” |
| activation\_link | a token, amivel a felhasználó aktiválni tudja a regisztrációját. |
| created | a létrehozás pontos dátuma |
| failed\_attempts | hibás jelszó próbálkozások |
| dropbox\_auth | amennyiben összekötötte dropbox-szal a fiókját, a token itt kerül eltárolásra |
| main\_folder | a felhasználó fő mappájának egyedi azonosítója; külső kulcs a folder táblából |

### File

A File tábla a felhasználók által feltöltött fájlokról tárol információkat.

Atttribútumok:

* id: a fájl egyedi azonosítója
* user\_id: a fájl tulajdonosa, külső kulcs a User táblából
* file\_name: a felhasználó által választott név
* system\_file\_name: a név ahogy a rendszeren van eltárolva
* created: a fájl létrehozásának dátuma
* public\_link: publikus fájl esetén egy egyedi token
* content: ha ez a fájl egy note akkor van kitöltve
* folder\_id: a fájlt tartalmazó mappa, külső kulcs a Folder mappából
* delete\_date: ha a fájlt kijelölte a tulajdonos törlésre, akkor ide bekerül a törlés ideje
* version: verzikövetés esetén a fájl verziószámát tárolja

### Folder

A mappákat reprezentálja és tárol róluk információt azok azonosítása és megtalálása érdekében.

Atttribútumok:

* id: a mappa egyedi azonosítója
* user\_id: a mappa tulajdonosa, külső kulcs a User táblából
* parent\_folder: a mappa szülő mappájának azonosítója
* folder\_name: a felhasználó által választott mappa név
* path: a mappa fizikai elérésének útvonala
* created: a mappa létrehozásának dátuma
* delete\_date: ha a mappát kijelölte a tulajdonos törlésre, akkor ide bekerül a törlés ideje

### File share

A fájlok megosztásának tárolásáért felelős tábla, melynek minden rekordja egy fájl hozzárendelésre egy új felhasználóhoz valamilyen jogosultsággal.

Atttribútumok:

* id: a megosztás egyedi azonosítója
* file\_id: a fájl azonosítója, külső kulcs a File táblából
* user\_id: a felhasználó azonosítója, aki kapja a jogosultságot a fájlra, külső kulcs a User táblából
* role\_id: a jogosultság azonosítója, külső kulcs a Role táblából
* created: a megosztás létrehozásának dátuma

### Role

A rendszerben definiált jogokat tárolja, amelyek segítségével fájlokhoz és note-okhoz hozzá lehet rendelni más felhasználókat. Ahogy a specifikációban említettem, három különböző jogosultságot különböztetünk meg: READ, WRITE, DELETE. Ezek a jogok az alkalmazás telepítésekor létre kell, hogy jöjjenek az adatbázisban a gondtalan működés elősegítésére.

Atttribútumok:

* id: egyedi azonosító
* name: a jogosultság szöveges neve
* priority: egy olyan érték, mely meghatározza az adott jogosultság erősségét a későbbi adatbázis műveletek könnyítésére

### Credential store

A program zavartalan működéséhez elengedhetetlen jelszavak tárolására lett létrehozva. A jelszavak az alkalmazás telepítésekor be kell, hogy kerüljenek a táblába, mert addig nem tud elindulni. A működéshez négy jelszóra van szükség:

1. SECRET\_KEY: a token kódolásához és dekódolásához szükséges, ezt az applikációnak is külön át kell adni
2. MAIL: az aktiváló- és reset email küldéséhez szükséges adatokat tartalmazza
3. DROPBOX\_KEY: az applikációhoz tartozó dropbox kulcs
4. DROPBOX\_SECRET: az applikációhoz tartozó dropbox secret

Atttribútumok:

* id: egyedi azonosító
* environment: szöveges név, ami leírja melyik ez a jelszó pontosan
* code: a jelszó maga

### Log

A felhasználói logokat tároló tábla, minden rekordja egy bejegyzés valamilyen felhasználó tevékenységről, fájlműveletről vagy mappaműveletről.

Atttribútumok:

* id: a log bejegyzés egyedi azonosítója
* user\_id: a felhasználó akihez a log tartozik, külső kulcsa a User táblából
* file\_id: a fájl amin műveletet végzett, külső kulcs a File táblából, ha az értéke nem null, akkor tudjuk, hogy fájlművelet volt
* folder\_id: a mappa amin műveletet végzett, külső kulcs a Folder táblából, ha az értéke nem null, akkor tudjuk, hogy mappát is érintett a művelet
* created: az esemény dátuma
* message: a szöveges bejegyzés

## Végpontok

A webes réteg úgynevezett route-okat vagy elérési pontokat tartalmaz, melyek az alkalmazás működéséshez szükséges funkciók elérését biztosítják. Kevés implementációs logikát tartalmaznak, fő feladatuk a felhasználóktól kapott adatok ellenőrzése, autentikáció és a modell megfelelő metódusainak meghívása. Az alkalmazás alapvetően négy külöböző típusú HTTP metódust használ:

* GET : adatok lekérdezése
* POST : új entitás létrehozása
* PUT : entitás módosítása
* DELETE : adatok törlése

### Autentikáció

Sikeres bejelentkezés esetén a rendszer visszaküld egy tokent, ami a kliens oldalon a sütik között kerül eltárolásra, majd később minden kéréssel a szerver felé kerül továbbításra a felhasználó azonosítása céljából.

Néhány route-ot leszámítva a szerver minden kérés előtt autentikálja a felhasználót, akitől a kérés érkezik. A kérésben a sütik között elküldésre kerül a token is, amit bejelentkezéskor a szerver visszaküldött a felhasználó felé. A szerver először megnézi, hogy ez a token szerepel-e a szerveroldalon tárolt tokenek között, ha igen, akkor dekódolja és ellenőrzi érvényességét. Ha minden rendben van, akkor továbbítja a kérést a megfelelő route-ra, ellenkező esetben a válaszban jelzi, hogy problémát talált.

Az felhasználó azonosítása a token segítségével történik, amelyből dekódolás után kiolvasható az egyedi azonosító.

### Adatok ellenőrzése

Minden route-ban specifikálva van, hogy milyen adatokat vár a felhasználótól. Egy példa a regisztrációt segítő elérési pontból:

{**'username'**: **None**,

**'password'**: **None**,

**'email'**: **"^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,4}$"**}

Vár egy felhasználónevet és egy jelszót, melyekre nincs megkötés, illetve egy e-mail címet, amire egy reguláris kifejezéssel megadott megkötés van. A felhasználótól kapott bemenetet és ezt a listát átadja egy validációs logikának, ami ellenőrzi, hogy minden paramétert átadott-e a felhasználó és a reguláris kifejezéseknek is megfelel-e. Azt is ellenőrzi, hogy a kapott érték ne legyen üres.

### UsersAPI

Olyan elérési pontokat tartalmaz, ami a felhasználókkal kapcsolatos műveleteket teszi lehetővé.

* /users/login: A felhasználók ezen keresztül tudnak bejelentkezni a rendszerbe. A body-ba belekerül a felhasználónév és jelszó.
* /users/register: Új fehasználókat lehet létrehozni ezen a route-on keresztül. A body-ba a felhasználónév, jelszó és e-mail kerül elküldésre a szerver felé. Sikeres regisztráció esetén kap a felhasználó egy aktiváló e-mailt a megadott címre.
* /users/activate/<token>: A felhasználó az aktivációs e-mailben kapott token segítségével itt tudja aktiválni a regisztrációját.
* /users/reset: Ha a felhasználó háromszor elrontja a jelszavát, akkor a rendszer kizárja és küld a megadott e-mail címre egy linket, amely segítségével új jelszót lehet megadni. A body-ban az emailben kapott token és az új jelszó szerepel. Siker esetén a felhasználó jelszava a megadottra változik.
* /users/delete: A felhasználók ezen keresztül tudják törölni a regisztrációjukat. Ez a törlés nem visszaállítható, minden korábban feltöltött fájl és adat törlődik a felhasználóval együtt.
* /users/change: E-mail cím és jelszó megváltoztatására van lehetőség ezen a route-on keresztül. A body-ba az új e-mail cím, a jelenlegi és az új jelszó kerül. Az aktuális jelszót mindenképpen meg kell adni, míg az új jelszó és e-mail opcionális és ennek megfelelően fog változni.
* /users/logout: Lehetőség van kijelentkezni ezen a route-on. A felhasználó egyedi tokene törlődik a sütik közül, valamint a szerveroldalon is.

### RolesAPI

A tárolt fájlok és note-ok megosztásához szükséges jogok kérdezhetők le.

* /roles: Ezen a route-on keresztül kérhetőek le a rendszerben található jogok.

### LogsAPI

Ezeken a route-okon keresztül lehet lekérni a felhasználókhoz tartozó logokat. Van lehetőség csak egy adott fájlra vagy mappára vonatkozó bejegyzéseket lekérni.

* /logs: Ezen a route-on keresztül tudják a felhasználók lekérni a hozzájuk tartozó bejegyzéseket. Minden olyan logot, ami tartalmazza a felhasználó azonosítóját visszaküldi a válaszban. Ezek lehetnek fájlokkal, mappákkal vagy a felhasználó tevékenységeivel (pl.: bejelentkezés) kapcsolatos műveletek.
* /logs/file/<file\_id>: A paraméterben megkapott fájlhoz tartozó összes bejegyzést elküldi a válaszban a felhasználónak. Az adott fájlhoz tartozó logok akkor kérhetőek le, ha a kérést küldő felhasználó a tulajdonos, vagy legalább READ joggal rendelkezik a fájlra.
* /logs/folder/<folder\_id>: A fájlokhoz hasonló, de itt a mappákhoz tartozó logok kérhetőek le. Mivel a mappák nem megoszthatóak, ezért ezekhez csak a mappa tulajdonosa fér hozzá.

### NotesAPI

Az alkalmazás képes note-okat is tárolni, melyek nem a fájlrendszerben, hanem az adatbázisban vannak tárolva, hiszen csak szöveges tartalomból állnak. Ezek a note-ok később törölhetőek, módosíthatóak és megoszthatóak is a route-okon keresztül.

* /notes: Az összes olyan note-ot visszaadja, amit a felhasználó hozott létre. A válaszba a note-ok tartalma is belekerül. Egy példa válasz JSON-ben:

{

"content": "test",

"created": "Wed, 01 Nov 2017 11:48:29 GMT",

"deleted": null,

"fileName": "test",

"folder": 1,

"id": 9,

"publicLink": null,

"version": 0

}

* /notes/note: Ezen a route-on keresztül tudnak a felhasználók új note-okat létrehozni. A kérésben el kell küldeni az új note nevét és tartalmát. Egy ilyen note maximum 300 karakter hosszú lehet, melyet a szerver a kérés fogadásakor ellenőriz.
* /notes/update: Lehetőség van a note-ok módosítására is, amelyet ez az elérési pont tesz lehetővé. A kérésben el kell küldeni a note azonosítóját valamint az új nevet és tartalmat. Egy note-ot a tulajdonosa vagy legalább a fájlhoz WRITE joggal hozzáférő felhasználó módosíthat. A karakterszámra itt is figyelni kell.
* /notes/<note\_id>: A felhasználók törölni is tudják a már nem használt note-okat, ezt tudják megtenni ezen a route-on keresztül. Egy note-ot csak a tulajdonosa vagy legalább DELETE joggal rendelkező felhasználó tud törölni. Ha egy olyan felhasználó törli a note-ot, aki nem a tulajdonosa csak hozzáférése van, akkor a tulajdonostól is törlődik.
* /notes/shared: Itt kérhetőek le azok a note-ok egy adott felhasználóhoz, amikhez csak hozzáféréssel rendelkezik, de nem ő a tulajdonosa.

### DropboxAPI

Ezeken a route-okon keresztül lehet műveleteket végezni a felhasználó Dropbox fiókjával, valamint a fiókok összekapcsolásához szükséges teendők is itt végezhetőek el.

* /dropbox (GET): Visszaadja az autentikációs URL-t, amire a felhasználónak fel kell navigálnia, és engedélyeznie kell az applikáció számára a hozzáférést a fiókjához. Sikeres engedélyezés után a felhasználó kap egy hozzáférési tokent.
* /dropbox (POST): A kapott hozzáférési tokent a felhasználó visszaküldi a szerver felé erre a route-ra.
* /dropbox/upload/<file\_id>: Ezen a route-on keresztül a paraméterben átadott fájlt a szerver felölti a felhasználó Dropbox fiókjára, amennyiben korábban összekapcsolta az applikációval. A fájl méretére nincs semmilyen megkötés. A fájlt mindig a felhasználó fő mappájába tölti fel.
* /dropbox/download: A felhasználó Dropbox fiókjáról letölti a body-ban megadott fájlt.

### FilesharesAPI

A fájlok megosztásához szükséges funkcionalitás érhető el ezeken a route-okon. A hozzáféréseket törölni is lehet utólag.

* /shares/share: Ezen az elérési ponton keresztül lehet jogokat adni más felhasználók számára. A kérés body-ban meg kell adni, hogy melyik fájlt kihez, és milyen hozzáféréssel szeretnénk hozzárendelni. Azt, hogy kinek ad hozzáférést, egy email címmel kell azonosítani.
* /shares/public/<file\_id>: Lehetőséget nyújt a paraméterben átadott fájl publikus megosztására. Ezek után a fájl bárki számára elérhető lesz a link ismeretében. Csak a fájl tulajdonosa publikálhatja a fájlt.
* /shares/private/<file\_id>: Az előző pontban bemutatott megosztást törli. Sikeres művelet után a fájl nem lesz többé elérhető a linken keresztül. Csak a fájl tulajdonosa tudja törölni a megosztást.
* /shares/revoke/<id>: Törli a paraméterben megadott megosztást. Ha sikeres a törlés, akkor azt követően már nem férhet hozzá a fájlhoz vagy note-hoz a felhasználó, aki számára meg volt osztva. Csak a fájl tulajdonosa törölheti.
* /shares/<file\_id>: A paraméterben megadott fájlhoz tartozó összes megosztást visszaadja. Csak a fájl tulajdonosa kérdezheti le.

### FilesAPI

A fájlokkal kapcsolatos műveletek végezhetőek el az alábbiakban bemutatott route-okon.

* /files/download/<file\_id>: Ezen az elérési ponton keresztül lehet letölteni a paraméterben megadott fájlt. A fájlokhoz csak akkor férhet hozzá a felhasználó, ha ő a tulajdonosa vagy legalább READ joggal rendelkezik.
* /files/download/folder/<folder\_id>: A paraméterben kapott mappát lehet innen letölteni, zip-be összetömörítve.
* /files/file/<folder\_id> (GET): Visszaadja a paraméterben megadott mappa tartalmát, hogy mely fájlokat tartalmazza. A mappa tulajdonosán kívül más nem tudja lekérdezni.
* /files/folder/<folder\_id> (GET): Az előző elérési ponthoz hasonlóan egy, a paraméterben megadott mappa tartalmát adja vissza, de nem a fájlokat, hanem az almappákat.
* /files/file/deleted: A felhasználó törölt fájljait adja vissza, melyek még nem lettek törölve a szerverről csak meg vannak jelölve, hogy 14 nap letelte után törlődjenek.
* /files/folder/deleted: A felhasználó törölt mappáit adja vissza. Ugyanazok a feltételek igazak rá, mint a törölt fájlokra.
* /files/file/<file\_id> (DELETE): A paraméterben megadott fájlt törli. A fájlt csak akkor tudja törölni a felhasználó, ha ő a tulajdonosa, vagy van legalább DELETE joga a fájlra.
* /files/folder/<folder\_id> (DELETE): Az előző route-hoz hasonlóan itt is törölni lehet, azonban itt egy mappának az azonosítóját kell átadni. Mivel mappát nem lehet megosztani, így csak a tulajdonos tudja törölni.
* /files/file/<folder\_id> (POST): Ez a route új fájl feltöltésére szolgál. Paraméterben adja át a felhasználó az új fájl kívánt helyét. A kérés elején a szerver ellenőrzi, hogy a feltöltendő fájlnak mekkora a mérete, ami, ha meghaladná az 1 GB-ot, akkor elutasítja a kérést. Azt is ellenőrzi, hogy a fájlt feltöltve meghaladja-e az engedélyezett felhasználói limitet, mely 1 GB-t.
* /files/createFolder: Ezen az elérési ponton keresztül lehet új mappákat létrehozni. A body-ba bekerül az új mappa neve, illetve a szülőjének az azonosítója.
* /files/getPublicFile/<link>: A publikussá tett fájlok ezen a route-on keresztül érhetőek el. A publikus fájlok akkor is letölthetőek, ha a kérést olyan személy indítja, akinek nincs regisztrációja.
* /files/search/<file\_name>: A keresés funkció elérését teszi lehetővé. Paraméterben vár egy fájl nevet a felhasználótól, ha talál a feltételeknek megfelelő egy vagy több fájlt, akkor azokat visszaküldi.
* /files/file/move: Ezen a route-on lehet fájlokat mozgatni más mappákba. A kérés body-ba a fájl azonosítója és az új mappa azonosítója kell, hogy bekerüljön. Sikeres művelet esetén visszaküldi a mozgatott fájl adatait az új szülőmappával.
* /files/folder/move: A fájl mozgatáshoz hasonló, azonban itt mappákat lehet áthelyezni máshova. A body-ba az áthelyezni kívánt mappa és az új szülőmappa azonosítója kell, hogy kerüljön. Siker esetén visszaadja a módosított mappa adatait.
* /files/file/rename: A fájlok átnevezését teszi lehetővé. Body-ban várja a fájl azonosítóját és az új nevet. Sikeres művelet esetén visszatér az új adatokkal.
* /files/folder/rename: Mappa átnevezésére jó, annyiban tér el a fájl átnevezésétől, hogy itt a body-ban a mappa azonosítóját várja.
* /files/folder/list: Visszaadja a felhasználóhoz tartozó összes mappát. A fő mappa Main néven található meg.
* /files/file/restore/<file\_id>: Azokat a fájlokat, amelyeket a tulajdonos vagy valamelyik DELETE joggal rendelkező felhasználó kijelölt törlésre, 14 napig vissza lehet állítani. Ez a route erre ad lehetőséget. Paraméterben kell átadni a fájl azonosítóját, melyet csak a tulajdonos tud visszaállítani.
* /files/folder/restore/<folder\_id>: Visszaállítja a törlésre kijelölt mappát a tartalmával együtt. Paraméterben kapja meg, hogy melyik mappáról van szó.
* /files/shared: A felhasználó számára lehetőséget nyújt lekérni azokat a fájlokat, amelyekhez van valamilyen jogosultsága. A saját fájljait nem adja vissza.

## Modell

A modell tartalmazza az adatbázis műveleteket és a tényleges logikát.

### DropboxModel

Kéréseket küld a Dropbox hivatalos API-ja felé, melyek segítségével fájlokat lehet le- és feltölteni, illetve segít összekapcsolni a felhasználó fiókjait. Az alábbi képen a két fiók összekötésének menete látható:



* auth\_url(): Visszaadja az URL-t, amelyre navigálva a felhasználó engedélyezheti az applikáció számára, hogy hozzáférjen a fájlokhoz.
* auth\_finish(token, user\_id): A felhasználótól kapott token-t ellenőrizve összeköti a két regisztrációt, és lementi adatbázisba a hozzáféréshez szükséges kódot, hogy azt ne kelljen minden alkalommal elkérni a felhasználótól.
* get\_access\_token(user\_id): A paraméterben kapott felhasználóhoz tartozó hozzáférési tokent adja vissza.
* upload\_file\_to\_dbx(user\_id, file\_id): Az adott felhasználóhoz tartozó fájlt feltölti Dropbox-ra is. Amennyiben van már ilyen nevű fájl, akkor azt felülírja. A limitált erőforrások miatt a fájl 100MB-os részletekben kerül feltöltésre.
* download\_from\_dbx(user\_id, input\_dictionary): Letölti a Dropbox-ról a felhasználó által megadott fájlt a kiválasztott mappába. Maximum 100MB-os fájl tölthető le, mert az API csak olyan végpontot biztosít ami a fájlt egyben tölti le. A fájl azonosítása után indít egy új szálat ahonnan a fájlt letölti, majd visszatér, hogy a letöltés kezdetét vette.

### CredentialstoreModel

Hozzáférést biztosít a credential\_store adatbázis táblához. A pogram indulásakor a táblában található összes jelszót betölti környezeti változóként a rendszer, hogy ne kelljen minden alkalommal adatbázis műveletet végezni, amikor például dekódol egy felhasználói tokent.

* get\_code(environment): A paraméterként átadott névhez tartozó jelszót adja vissza.

### UsersModel

A felhasználókkal kapcsolatos műveletek logikáját tartalmazza és az adatbázis elérést.

* login\_user(username, password, ip): Megkeresi az adatbázisban a kapott felhasználónévhez tartozó rekordot. Amennyiben létezik, dekódolja a tárolt jelszót, és összehasonlítja a felhasználótól kapottal. A művelet sikerességét logolja, majd visszatér egy új tokennel, ha a megadott jelszó jó volt.
* increment\_bad\_password(user): Amennyiben a felhasználó által megadott jelszó rossz volt, az adatbázisban növeli a hibás próbálkozások számát. Ha ez a szám eléri a hármat, akkor kizárja a rendszerből és a megadott e-mail címre küld egy levelet, aminek a segítségével új jelszót lehet beállítani.
* register\_user(username, user\_password, email): Készít egy új rekordot a User táblában a megadott adatokkal. A jelszót először hash-eli, majd a kapott értéket tárolja el.
* activate\_user(token): A paraméterben kapott token alapján megkeresi az adatbázisban a rekordot és aktiválja azt az activation\_link attribútum null értékbe állításával. Készít egy új mappát az aktivált felhasználó számára, ahova a fájljai majd feltöltésre kerülnek.
* reset\_user(token, password): A kapott token alapján azonosítja a rekordot, ahova az új jelszót be kell állítania.
* delete\_user(user\_id): Törli a paraméterben megkapott azonosítójú felhasználót az adatbázisból. Törlés után nem lehet már visszállítani. A felhasználó valamennyi fájlja törlődik.
* change\_user\_data(user\_id, input\_dictionary): Lehetőséget nyújt a felhasználók számára, hogy megváltoztassák az e-mail címüket és a jelszavukat. A paraméterben kapott map-ben megnézi, hogy van-e új jelszó vagy e-mail cím, és ha van, akkor azt frissíti az adatbázisban.
* get\_user\_data(user\_id): Visszaad egy user objektumot a paraméterben kapott azonosító alapján.

### RolesModel

A jogosultságok lekérdezéséhez szükséges metódus megvalósítását tartalmazza. Új jogosultság felvételére nincs lehetőség, mivel az alkalmazás működése nem igényli, hisz nincs admin szerepkör.

* get\_all\_roles(): Visszaadja az adatbázisban található összes jogosultságot.

### NotesModel

A felhasználói note-okon végzett műveletek találhatóak benne.

* create\_note(user\_id, input\_dictionary): Létrehoz a paraméterben kapott felhasználónak egy új note-ot. A note címe és tartalma az input\_dictionary-ben található.
* delete\_note(user\_id, note\_id): Törli a paraméterben meghatározott note-ot, amennyiben a user\_id olyan felhasználót jelöl, aki vagy a tulajdonosa a note-nak, vagy pedig van rá legalább DELETE jogosultsága.
* update\_note(user\_id, input\_dictionary): Egy note címét és tartalmát lehet módosítani a segítségével, ha a paraméterben átadott felhasználónak legalább WRITE jogosultságva van, vagy ő a tulajdonos.
* get\_all\_notes(user\_id): Visszaadja a felhasználóhoz tartozó valamennyi note-ot.
* get\_shared\_with\_me\_notes(user\_id): A felhasználóval megosztott note-okat lehet lekérdezi ebben a metódusban. A függvény visszaad egy note-ot, akármilyen jogosultsága is van rá.

### LogsModel

A logoláshoz szükséges műveleteket foglalja magába.

* create\_log\_entry(user\_id, message, file\_id, folder\_id, session): Létrehoz egy új bejegyzést a a kapott adatokkal. Ha a fájl vagy mappa azonosító üres akkor NULL érték kerül az adatbázisba.
* get\_user\_entries(user\_id): Visszaadja a paraméterben kapott felhasnálóhoz tartozó valamenyi log bejegyzést.
* get\_file\_entries(user\_id, file\_id): Az előzőhöz hasonló, azonban itt egy konkrét fájlhoz tartozó bejegyzéseket ad csak vissza.
* get\_folder\_entries(user\_id, folder\_id): A paraméterben kapott mappa összes bejegyzését visszaadja.

### FilesModel

A fájlokkal végzett műveletek, letöltés és feltöltés megvalósítása.

* allowed\_file(filename): Ellenőrzi a fájlnév alapján, hogy a fájl nincs-e a nem megengedett kiterjesztések között.
* get\_all\_files(user\_id, folder\_id): A paraméterben kapott mappában található összes fájlt visszaadja, ha azok nincsenek törölve.
* get\_all\_folders(user\_id, folder\_id): Az előző függvényhez hasonló, azonban itt a paraméterben kapott mappában található mappákat adja vissza.
* get\_all\_deleted\_files(user\_id): A felhasználóhoz tartozó összes törölt fájlt visszaadja, hierarchiától függetlenül.
* get\_all\_deleted\_folders(user\_id): A törölt mappákat adja, vissza amelyek a a felhasználóhoz tartoznak. A hiearchia itt nem számít.
* search\_user\_file(user\_id, file\_name): A kapott fájlnév részlet alapján lekérdezi a felhasználóhoz tartozó olyan fájlokat, amelyeknek nevében benne van a kapott név részlet. Kis és nagybetű között nincs különbség a keresés során.
* upload\_file(user, folder\_id): Egy új fájl feltöltésére szolgál, először megkeresi a mappát, ahova tölteni kell a fájlt, létrehoz egy teljesen véletlen karakterekből álló nevet a fájlnak, majd létrehoz egy stream-et amin keresztül a fájl részletekben felöltésre kerül. Erre azért van szükség, hogy a Raspberry PI eszköz memória használata ne érje el a fizikai határait.
* create\_file(user\_id, filename, sys\_fname, folder\_id): Az adatbázisban létrehoz egy új rekordot az éppen feltöltésre váró fájlhoz.. Ellenőrzi, hogy van-e már ilyen nevű fájl az adott mappában, ha igen akkor a verziószámot növeli eggyel.
* remove\_file(user\_id, file\_id): Kijelöli törlésre a paraméterben kapott fájlt, azonban az 14 napig még helyreállítható lesz. Törlődik azonban a fájlhoz tartozó összes megosztás és ezek visszaállítás esetén már nem jönnek vissza. Ha van ugyanilyen nevű fájl nagyobb verziószámmal a mappában, akkor azoknak csökkenti azt eggyel.
* remove\_folder(user\_id, folder\_id): Az adott mappát kijelöli törlésre úgy, hogy a tartalma is törölve lesz, de 14 napig még visszaállíthatóak. Mappák esetén nincs verziókövetés, továbbá a kijelölt fájlok verziójával nem kell törődni, hisz ugyanabban a mappában vannak.
* delete\_shares(user\_id, file\_id): Törli a fájlhoz tartozó összes megosztást,melyek nem visszaállíthatóak.
* crt\_folder(user\_id, input\_dictionary): Létrehoz egy új mappát, úgy, hogy előtte ellenőrzi a szülőmappa tartalmát nehogy azonos nevű legyen egy másikkal. Az új mappa neve nem lehet „…” és üres sem. Fizikailag is létrejön egy új mappa a hierarchiának megfelelően.
* rename\_file(user\_id, input\_dictionary): Átnevezi a fájlt a paraméterben kapott adatok szerint, közben figyelve arra, hogy a fájl régi nevével azonos nevű fájlok verzió változzanak és a szóban forgó fájl pedig új verziót kapjon az új név alapján.
* rename\_folder(user\_id, input\_dictionary): Átnevezi a fájlt fizikailag és az adatbázisban is, arra figyelve, hogy azonos testvér mappája ne legyen. Frissíteni kell a hierachiában alatta található összes mappa elérési útvonalát is.
* move\_folder(user\_id, input\_dictionary): Áthelyezi a paraméterben kapott mappát a kijelölt helyre, fizikailag és az adatbázisban is, ha ott nincs ilyen nevű mappa. Figyelni kell a hierachiában alatta álló mappák elérési útvonalának megváltoztatására.
* move\_file(user\_id, input\_dictionary): Áthelyezi a adott fájlt a kívánt helyre mind az adatbázisben mind fizikailag, ügyelve arra, hogy a verziók az új és régi mappán belül megfelelően változzanak.
* delete\_job(): Ez egy időzített művelet, amely külön szálon fut, úgy ,hogy a fő szálat nem várakoztatja futás közben. Alapvetően 24 óránként fut le, azonban ez az érték könnyen változtatható. Az adatbázisból lekérdezi az összes olyan mappát és fájlt aminek a törlési ideje legalább 14 nap, majd az adatbázisból és fizikailag is törli ezeket. Ez a művelet már nem visszafordítható, tehát az így törölt fájlok végleg elvesztek.
* get\_file\_data(user\_id, file\_id): A paraméterben kapott fájl letöltéséhez szükséges adatokat adja vissza, amelyek a fájl fizikai elérési útvonala, a valódi fájlnév és feltöltés során hozzárendelt fájlnév.
* get\_folder\_data(user\_id, folder\_id): Előkészíti a mappát letöltéshez, majd visszaadja az adatokat hozzá. Először átmásolja a mappát a zip könyvtárba, törli belőle a törlésre kijelölt mappákat és fájlokat, majd átnevezi a fájlokat, hogy azoknak a felhasználó által megadott nevük legyen. Ezek után tömöríti a mappát és törli az átmásolt mappát, hogy ne foglaljon több helyet. Visszatérési értékként átadja az adatokat, amelyek ahhoz kellenek, hogy a zip-et le lehessen tölteni.
* get\_public\_file(public\_link): A paraméterként kapott token alapján azonosítja a fájlt, majd visszaadja az adatokat amik a letöltéshez szükségesek.
* get\_folder\_list(user\_id, file\_id): A felhasználóhoz tartozó valamennyi mappát visszaadja, viszont egy adott mappa nevében a teljes hierarchia benne lesz.
* restore\_file(user\_id, file\_id): Egy törlésre kijelölt fájlt visszaállít eredeti helyére, figyelve arra, hogy a verziószáma nagyobb legyen eggyel mint az aktuális legnagyobb.
* get\_parent\_folder(user\_id, folder\_id): Visszaadja a paraméterben kapott mappához tartozó szülőmappa adatait.
* restore\_folder(user\_id, folder\_id): Visszaállítja a törlésre kijelölt mappát és tartalmát az eredeti helyére, ha azóta nem lett létrehozva ott azonos nevű mappa.
* get\_shared\_with\_user\_files(user\_id): Az adott felhasználóhoz valamilyen jogosultsággal hozzárendelt fájlok listáját adja vissza.

### FilesharesModel

A fájlok megosztásával kapcsolatos függvények találhatóak benne.

* public\_file(user\_id, file\_id): A paraméterben megkapott azonosítójú fájlt publikussá teszi, úgy, hogy a fájlhoz tartozó összes megosztást törli először. A fájlhoz generál egy tokent, amely egyedi azonosítóként fog szolgálni és ennek a segítségével lehet majd letölteni.
* revoke\_public(user\_id, file\_id): Paraméterben kap egy fájlt, amely ha publikus akkor azt újra priváttá teszi, ezek után a fájlt már nem lehet letölteni az addigi tokennel.
* share\_file(user\_id, input\_dictionary): Létrehoz egy új megosztást az input\_dictionary-ben definiált adatokkal. Meg kell adni a jogosultságot, a felhasználót akit meghatalmazunk és fájl azonosítóját. Egy fájl csak akkor osztható meg, ha nem publikus.
* delete\_share(user\_id, share\_id): Törli a paraméterben megadott megosztást.
* get\_shares(user\_id, file\_id): Visszadja, hogy az adott fájlhoz milyen megosztások tartoznak. Csak a fájl tulajdonosa kérheti le.

### TokensModel

Az azonosítást elősegítő tokenen végzett műveletek megvalósítását tartalmazza.

* encode\_token(id): Készít egy új tokent, amit az alábbi adatok kódolásával hoz létre:

payload = {  
 **'exp'**: datetime.datetime.utcnow() + datetime.timedelta(minutes=60),  
 **'user'**: id  
}

Belekódolja a felhasználó egyedi azonsoítóját, illetve, hogy meddig érvényes.

* decode\_token(token): Dekódolja a kapott tokent és megállapítja érvényességét.
* login\_required(f): Megvizsgálja, hogy melyik route-ra érkezett a kérés és amennyiben autentikáció szükséges, akkor a tokent átadja a fentebb említett függvénynek.

### Felhasználói limitek

Korábban említettem, hogy a rendszer jelenleg limitekkel rendelkezik köszönhetően annak, hogy kevés fizikai memóriával és tényleges tárhellyel rendelkezik a futtató eszköz.

* maximum 1GB-os fájlokat lehet feltölteni annak biztosítása érdekében, hogy több felhasználó is tudja használni az alkalmazást
* a note-ok maximum 300 karakterből állhatnak
* Dropboxról maximum 100MB-os fájl tölthető le, mert csak egyben tölthető le a fájl

Ezek betartásáért létrehoztam két függvényt, amelyek a megfelelő route-ok meghívása előtt fognak lefutni és ellenőrzik a paramétereket:

* limit\_content\_length(max\_length): Megnézi a kérés méretét, és ha az nagyobb mint a paraméterben kapott maximális méret, akkor a kérést elutasítja.
* user\_file\_limit(): Ellenőrzi, hogy ha a kérésben kapott fájlt is feltölti a felhasználó vagy létrehozza a note-ot akkor nem lesz-e nagyobb az össz fájlmérete mint a megengedett. Amennyiben igen, a kérést elutasítja.

### Logolás

Még nem esett szó a szerver oldalon történő hibák logolásáról, amik a fejlesztés során és a későbbi használat alatt is rendkívül hasznosak tudnak lenni. Mivel az alkalmazás nem igényel komplexebb logolást ezért amellett döntöttem, hogy saját magam által megírt loggert fogok használni. Bevezettem egy osztályt, ami három szintet tartalmaz magában:

1. Info
2. Warning
3. Error

Az egyes logüzenetek átadása során megadható a szintje, ezzel jelezve a fontosságát. A rendszer indulása során be van állítva az a logolási szint, amely legalább szükséges ahhoz, hogy az üzenetet a rendszer kiírja a logfájlba.

A Flask könyvtár lehetőséget biztosít arra, hogy az alkalmazásban feliratkozzak nem elkapott hibákra vagy, ha a szerver 500-as HTTP kóddal tér vissza, tehát valamilyen hiba történt a kérés kiszolgálása közben. Ebben az esetben as elkapott eseményt Error szinttel adom át a logolást végző függvénynek.

## UI

Ebben a fejezetben bemutatásra kerül a felhasználói felület felépítése és működése képekkel illusztrálva.

### Művelet sikeressége

A frontend normál működése során sok kérést küld a backend felé és ezeknek a kéréseknek a sikerességéről tájékoztatni kell a felhasználót is, hogy tudja, hogy az egyes műveletek valóban sikerültek-e. Egy adott művelet után a felhasználó a jobb felső sarokban egy felugró kis zöld ablakot láthat benne egy rövid üzenettel, hogy sikerült a módosítás. Míg sikertelen művelet esetán egy piros ablakot láthat, amely a művelet sikertelenségéről tájékoztat.

### Bejelentkezés

A helyes URL beírása után a felhasználók erre az oldalra jutnak először. Ha már van regisztrációjuk korábbról, akkor a felhasználónevük és a jelszavuk megadásával tudnak belépni a rendszerbe. A kérést csak akkor küldi el a szerver felé a kliens, ha mindkét mező ki lett töltve, hiányzó adat esetén az oldal figyelmezteti a felhasználót. A folyamat az Enter billentyű lenyomásával vagy a „Login” gombra kattintva indítható.



### Regisztráció

Ha a felhasználónak még nincs regisztrációja akkor a bejelentkező oldalon a „Create an account” feliratra kattintva átnavigálhatnak a regisztrációs oldalra. Itt egy egyedi felhasználónevet és e-mail címet kell megadniuk, továbbá egy választott jelszót. Az oldal ellenőrzi,hogy minden mező ki lett-e töltve, ha igen akkor validálja a bemenetet és ha mindent rendben talál, akkor küldi csak el a szerver felé a kérést. Hiány vagy helytelen adat esetén az oldal figyelmezteti a felhasználót. A folyamat az Enter billentyű lenyomásával vagy a „Register” gombra kattintva is elindítható.



### Files menü

Belépés után a menüsor segítségével tud a felhasználó navigálni az alkalmazáson belül. A menü három lenyíló ablakot tartalmaz, az első a „Files” ahol a fájlokkal kapcsolatos dolgok érhetőek el.



A „My Files” menüpontra kattintva a felhasználó a saját fájljait és mappáit tekintheti meg listába rendezve, ahol felül először a mappák alattuk pedig a fájlok láthatóak. A hierarchiában lehetőség van lépkedni, a kívánt mappára duplán kattintva, ha felhasználó nem a legfelső fő mappájában van, akkor egy „…” mappa jelöli az előző mappába történő visszajutást. Fájlok kereséséhez a felső mezőbe kell beírni a keresendő szót. A kereső nem tesz különbséget kis és nagybetű között és mindegy egyes változást érzékelve új kérést küld a szerver felé, tehát a keresés folytonosan történik.



A listában található fájlok és mappák kijelölhetők, amely segítségével megtekinthetőek a részleteik. Egy fájlra kattintva a képernyő jobb oldalon felugrik egy ablak az következő információkkal: név, feltöltés dátuma, verziószám, törlés dátuma és a fájlra vonatkozó logok. Mivel a törölt fájlok oldalán is van lehetőség megtekinteni az ottani fájlok részleteit, ezért a két oldal ugyanazt a komponenst használja erre a funkcióra. Az ablakon lehetőség nyílik műveleteket végezni a fájlon:

* A név mező átírása után a „Rename” gombra kattintva a fájl átnevezhető.
* A „Delete” gombra kattintva a fájl törölhető, ha az még nincs törölve.
* A „Move” gomb mellett a lenyíló listából választható egy új mappa, majd a gombra kattintva a fájl áthelyezhető oda.
* A „Share” gombra kattintva a felugrik egy új ablak, ahol más felhasználókkal lehet megosztani a fájlt.
* A „Restore” gomb segítségével vissza lehet állítani a fájlt amennyiben az törölve volt.
* A „Download” gomb elindítja a fájl letöltését.
* A „Dropbox” gomb feltölti a felhasználó Dropbox fiókjára a fájlt ha össze van kötve az alkalmazással.



Egy kiválasztott fájl megosztásait tehát a „Share” gomb megnyomásával tudjuk kezelni. Az így felugró ablakon látehatjuk a kiválasztott fájl nevét és ha publikus a fájl akkor a token-t, amely segítségével letölthető. Új jogok úgy adhatóak hozzá, hogy először kiválasztunk egy jogosultságot, megadjuk a felhasználót akinek szánjuk majd rákattintunk a „Share” gombra az ablakon. Ha a fájl nem publikus akkor a „Public” gombra kattintva tehetjük azzá, visszafelé pedig a „Revoke Public” gombbal. A gombok alatt található egy lista, ha a fájl meg van osztva legalább egy felhasználóval, ebben a listában a felhasználó e-mail címe és a jogosultság típusa látható, illetve egy kék „x” amely a jogosultság törlésére szolgál.



A fájlokhoz hasonlóan tehát megtekinthetőek a mappák részletei is mint a név, létrehozás dátuma, törlés dátuma és a mappához tartozó logok. Itt is fenn áll az, hogy ahol a törölt mappák vannak megjelenítve az az oldal ugyanezt a komponenst használja és a gombok ennek megfelelően vannak engedélyezve és tiltva. Az itteni a gombok ugyanazt a funkciót jelölik, mint a fájlokhoz tartozó komponensen.



A keresést segítő mezőtől jobbra három ikon található, egy mappa, egy fájl és egy Dropox ikon.



A Dropbox ikonra kattintva egy felugró abalkon lehet megadni egy olyan Dropboxon tárolt fájl adatait, amelyet szeretne a felhasználó letölteni onnan. A „Path” mezőbe kell megadni a fájl elérését, míg a „Filename” mezőbe a a fájl nevét. Helyes adatok esetén a fájl átkerül ide is.



A mappa ikonra kattintva a felugró ablak segítségével egy új mappa hozható létre, ott ahol a felhasználó éppen tartózkodik.



Míg a fájl ikonra kattintva lehet feltölteni új fájlokat a felugró ablakon keresztül, szintén oda, ahol a felhasználó éppen tartózkodik.



A „Files” menü második pontja a „Shared with me” menüpont, ahol a felhasználó azokat a fájlokat látja amik meg lettek osztva vele. Az egyes fájlok kiválaszthatók és műveletek végezhetőek rajtuk a jogosultságnak megfelelően. Törölni például csak akkor tudja ha DELETE jogosultsággal rendelkezik.



Az utolsó menüpont a lenyíló listában a „Deleted”, ide kerülnek azok a fájlok és mappák, amiket a felhasználó kijelölt törlésre, azonban a 14 nap még nem telt le. Az egyes elemek kijelölésekor megjelenik a korábban bemutatott ablak, amely a fájl részleteit tartalmazza. Azonban most a törlést jelölő dátum ki van töltve. A fájl vagy mappa a „Restore” gomb megnyomásával visszaállítható korábbi helyére. A 14 nap letelte után a fájl vagy mappa véglegesen törlődik a rendszerből és nem visszaállítható sehogyan sem.



### Notes menü

A menüsor második a eleme a „Notes” ahol a note-okkal kapcsolatos menüpontok kerültek.



Az első menüpont a „My Notes” ahol a felhasználó a saját note-jain tud műveletek végezni vagy teljesen újat létrehozni. Felül egy listát láthat a note-jaiból, amelyben az egyes elemek kiválaszthatóak. Kiválasztás után az adott note adatai jelennek meg. A lehetséges műveletek:

* Ha még nincs kiválasztva egy note sem akkor a a név és tartalom megadása után a „New” gombra kattintva létrejön az új note. Amennyiben ki van választva egy már létező, akkor a „New” gombra kattintva törlődik a kiválasztás és meg lehet adni az új note adatait.
* Az „Update” gomb egy kiválasztott note adatain módosít megnyomás után.
* A „Delete” gomb segítségével lehet törölni a note-okat.
* A „Share” gomb megnyomásával pedig a már korábban bemutatott megosztást segítő ablak ugrik fel.



A második menüpont a „Shared with me” ahol a felhasnálóval megosztott note-ok tekinthetőek meg. Az oldal kialakítása hasonló az előzőhöz azonban itt csak két műveletre van lehetőség. Az egyik a módosítás az „Update” gombbal a másik pedig a törlés a „Delete” gombbal. A felhasználó az adott műveletet csak akkor tudja végrehajtani, ha van hozzá megfelelő jogosultsága.



### User menü

A menüsor utolsó eleme a „User” menü, itt a felhasználóval kapcsolatos műveletek végezhetőek el.



Az első menüpont a „Change Data”, amelyre kattintva a felhasználó a meg tudja változtatni az e-mail címét és/vagy a a jelszavát. A jelenlegi jeslzavát mindenképp meg kell adnia helyesen ahhoz, hogy a művelet sikeres legyen.



A következő menüpont a „Dropbox” ahol a felhasználó össze tudja kötni a Dropboxos és az itteni fiókját. Az oldalon egy linket lát ahova átnavigálva, be kell lépni a Dropboxos regisztrációjába, majd ott engedélyezni tudja az alkalmazás számára a hozzáférést. A művelet végén kap egy hozzáférési tokent, amelyet az oldalra be kell másolnia és megnyomnia az „Authorize” gombot, ezek után a két regisztráció össze lesz kötve.



A harmadik menüpont a lenyíló listában a „Logs” ahol az összes felhasználóhoz kapcsolódó log bejegyzés megtekinthető. Itt lehetnek sikeres és sikertelen bejelentkezési kísérletek vagy a felhasználóhoz tartozó mappákon és fájlokon végzett műveletek. A bejegyzések nem módosíthatóak és nem törölhetőek.



Az utolsó előtti menüpont a „Logout”, amelyre kattintva a felhasználó kijelentkezik a rendszerből és visszakerül a bejelentkező oldalra. Az utolsó pedig a „Delete Account”, ahol törölni törölni tudja a regisztrációját. Kattintás után felugrik egy megerősítő ablak ahol az „OK”-ra kattintva a felhasználó fiókja törlődik véglegesen, tehát a művelet nem visszaállítható.



# Mérések

A fejezetben olyan mérések eredményét és összehasonlítását fogom bemutatni, amelyeket először elvégeztem a saját gépemen, ahol a fejlesztés történt majd a Raspberry PI eszközön is. A kéréseket egy harmadik külön gépről indítottam. A cél bemutatni, hogy milyen teljesítmény csökenéssel jár a kérések kiszolgálása a lassabb eszközön. Az alkalmazás Windows és Raspbian operációs rendszeren is tud futni.

Az erősebb gép lényeges specifikációi:

* RAM: 16GB
* Tárhely: SSD 256GB, 1TB HDD (Az adatbázis műveletek HDD-re íródnak, a fájlműveletek az SSD-n történnek)
* Processzor: i7 6700 CPU @ 3.40 GHz
* Internet sebessége: 1Gbit (A Raspberry PI eszköz specifikáció szerint 100Mbit-re képes Ethernet kábelen keresztül, mivel nem USB-t használok.)
* Windows 10 64-bit

Amiről még szót kell ejteni hogy az RPI eszköznek mik a pontos írási és olvasási limitei. Hálózat esetén tehát 100Mbit-re képes ami átszámítva 12.5 MB/s. Az SD kártya írási és olvasási sebességét az alábbi parancsokkal teszteltem:

Írás: dd if=/dev/zero of=~/test.tmp bs=500K count=1024

Olvasás: dd if=~/test.tmp of=/dev/null bs=500K count=1024

A kapott érték írásra 16.8MB/s, míg olvasásra 22.6MB/s, tehát mindkét esetben a hálózat az ami korlátozni fog pár fájlműveletet.

1. 1000 egymás után indított kérés:

A kiválasztott route /files/folder/<folder\_id>, amelyet azért választottam, mert van benne adatbázis elérés így jobb képet kapunk. Arra ügyeltem, hogy az adatbázisban található rekordok azonosak legyenek, tehát azonos a két mappa tartalma. A kapott eredmények átlagolva:

PC: ~14 s

RPI: ~36 s

1. 1000 kérés párhuzamosan:

A route ugyanaz mint az előző mérés során a változás annyi, hogy a kérések több szálon futnak a szerver felé. Összesen 3 szálat használtam és mindegyik szál pontosan 333 darab kérést küld.

PC: ~6 s

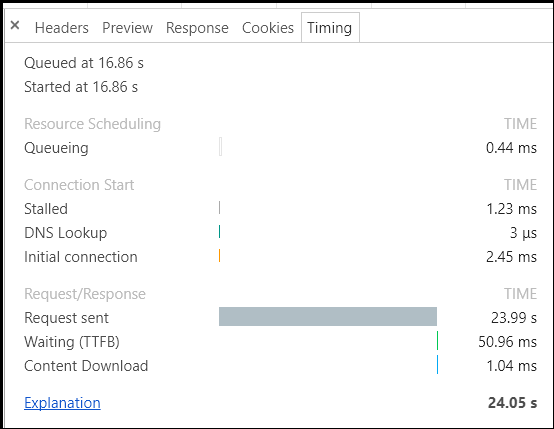
RPI: ~33 s

Jól látható javulást értünk el PC esetén, azonban az RPI szinte semmilyen pozitív elváltozást nem mutat.

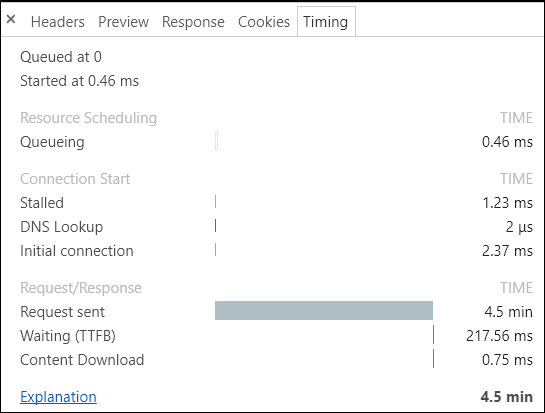
1. 1 GB-os fájl feltöltése majd letöltése:

A fájl pontos mérete (1075MB).

Ebben a mérésben ugyanazt a fájlt feltöltöttem mindkét rendszerre. A PC esetén 24 s alatt töltődött fel a fájl, ami ~45MB/s-es feltöltési sebességet jelent.



Az RPI rendszerre pedig 4.5 perc alatt sikerült feltölteni ami 270 másodperc. Ezt átszámolva ~4MB/s-es feltöltési sebességet kapunk, jól látható tehát, hogy nem közelíti meg a fizikai limitet.



1. 3 GB-os fájl feltöltése majd letöltése:
2. 2 darab 1GB-os fájl feltöltése majd letöltése egy időben:

# Összefoglalás

A projekt tervezése és fejlesztése alatt megismerkedtem új technológiákkal, így sokat tanultam és fejlődtem szakmailag. A feladat megoldása során több problémába is ütköztem, ahol mérlegelnem kellett több lehetséges megoldást, majd kiválasztani a szerintem legjobbat.

A project végére elkészítettem egy jól működő fájlmegosztó portált, amely egy egyszerű felhasználó számára minden igényt kielégít ahhoz, hogy fájljait ott tárolja és megossza másokkal. A szoftvert összekötöttem egy másik népszerű portállal, úgy ,hogy lehessen oda feltölteni fájlokat, illetve onnan letölteni.

Mesimerkedtem a Raspberry PI eszközzel és sikeresen telepítettem rá az elkészült softvert. Méréseket végeztem, hogy átfogó képet kapjak arról mekkora lassulással jár, ha az alkalmazás az RPI-n fut, nem pedig egy jóval erősebb gépen.

## Továbbfejlesztés

Ebben a fejezetben azok a még meg nem valósított funkciók kerülnek bemutatásra, amelyek az elkészült alkalmazást még élvezhetőbbé és kényelmesebbé tehetik.

* Cache menedzsment: Az eszköz memória menedzsmentje jelenleg úgy működik, hogy, ha egy fájl feltöltésre kerül a rendszerre akkor azt a cache-be eltárolja addig amíg tudja, majd mindig felszabadít annyit, hogy a következő beérkező részletet is be tudja tenni.

A memória állapota az operációs rendszer indítása után:



A memória állapota a backend indítása után:



A memória állapota a frontend-es Lite server indítása után:



A memória állapota miközben egy 3 GB-os fájlt töltök fel:



A képen jól látható, hogy a tényleges szabad memória közel ~600MB (második sor free oszlop), azonban a cached memória ~550MB. A cél az lenne, hogy az eszköznek ne kelljen a cache-elt memóriából kijelölnie új területet, hanem a szoftver programozottan ürítené azt, majd megmérni, hogy milyen gyorsulás érhető el így.

* Kérés menedzsment: Ha az eszköznek sok kérést kell kiszolgálni például sok fájl feltöltése, akkor előfordulhat hogy ténylegesen elfogy a memória ami a rendszer összeomlásához vezethet. A terv az lenne, hogy egy feltöltés, letöltés vagy dropboxos művelet elkezdése előtt a szoftver ellenőrízné a szabad memóriát és csak akkor engedélyezi, ha van legalább ~150MB szabad.
* Mappa megosztás: A rendszerben jelenleg csak fájlokat lehet megosztani, azonban a felhasználói élményt nagyban javítaná, ha hasonló feltételekkel mappákat is meg lehetne osztani.
* Tárhelybővítés: Az eszköz jelenleg egy 16 GB-os microSD kártáyval rendelkezik csak, amely sajnos kis mozgásteret ad. Szeretném kibővíteni és megszűntetni a rendszerben található felhasználói limiteket.
* Nginx: Nginx webszerver feltelepítése az eszközre és bekonfigurálása, hogy a be és kimenő forgalom csak rajta menjen keresztül.
* Https: Jelenleg nincs titkosítás a frontend és a backend kommunikációja között ezért nem nevezhető biztonságosnak. Https bevezetésével szeretném ezt a problémát kiküszöbölni a későbbiekben.
* Fájl titkosítás: A feltöltött fájlok jelenleg nincsenek titkosítva szerveroldalon. Biztonsági okokból viszont jó lenne ennek a megvalósítása.

# Irodalomjegyzék

|  |  |
| --- | --- |
| [1] | „Flask,” [Online]. Available: http://flask.pocoo.org/docs/0.12/. [Hozzáférés dátuma: 21 november 2017]. |
| [2] | „Raspbian OS,” [Online]. Available: http://www.raspbian.org/. [Hozzáférés dátuma: 6 November 2017]. |
| [3] | „Dropbox Python API,” [Online]. Available: http://dropbox-sdk-python.readthedocs.io/en/latest/. [Hozzáférés dátuma: 6 november 2017]. |