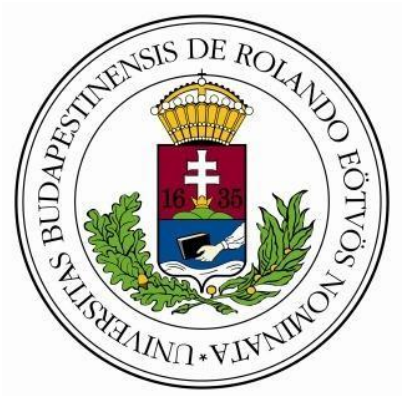


Eötvös Loránd Tudományegyetem

Informatikai Kar

Programozáselmélet és Szoftvertechnológia tanszék

HomeCloud

Témavezető:

Dr. Szendrei Rudolf

adjunktus, Ph.D.

Szerző:

Novák Márk Attila

Programtervező informatikus BSc.

Budapest, 2021

Tartalomjegyzék

1. Bevezetés 1
2. Felhasználói dokumentáció – HomeCloud 3
2.1. A megoldandó feladat 3
2.2. Rendszerkövetelmények 3
2.3. Telepítés 3
2.4. Eltávolítás 5
2.5. Az első indítás előtti előkészületek 5
2.6. Az alkalmazás használata futás közben 6
3. Felhasználói dokumentáció – HomeCloudMobile 11
3.1. Rendszerkövetelmények 11
3.2. Telepítés 11
3.3. Az alkalmazás használata futás közben 12
3.4. Státusz üzenetek 16
4. Fejlesztői dokumentáció 19
4.1. Fejlesztőeszközök 19
4.2. A fejlesztés előkészületei 20
4.3. Az alkalmazás osztályai 21
4.4. A főmenü 21
4.4.1. A MainMenuController osztály 22
4.5. Help ablak 24
4.6. Network Discovery 24
4.6.1. A HomeCloudNetworking osztály 25
4.7. File transfer 28
4.7.1. A FileTransferController osztály 29
4.7.2. A FileTransferUISetter interfész a FileTransferController osztályban 31
4.7.3. A Client osztály 32
4.7.4. A Server osztály 33
4.8. Directory listener 35
4.8.1. A WatchDirectoryController osztály 36
4.8.2. A WatchDirectoryUIsetter interfész a WatchDirectoryController osztályban 37

4.8.3. A MessageQueue osztály 38
4.8.4. A WatchHandler osztály 39
4.8.5. A WatchListener osztály 40
4.8.6. A WatchDir osztály 42
5. Fejlesztői dokumentáció – HomeCloudMobile 46
5.1. Fejlesztőeszközök 46
5.2. A főmenü 46
5.3. Network Discovery 48
5.4. Watch a directory és File transfer 48
6. Tesztelés 51
7. Összegzés 54
8. Irodalomjegyzék 55

1. Bevezetés

Napjaink modern világában az adat talán az egyik legértékesebb dolog. Azonban az adat szinte értéktelen, ha nem lehet hozzáférni és szállítani. Ugyanakkor, nem csak tárolni, hanem hordozni is kell, amelyre az évek során számos megoldás született, például: floppy, cd, dvd, pendrive. Valószínűleg az utolsó az egyik legnépszerűbb adathordozó egység manapság. Egy hátránya van azonban, ha nincs nálunk, akkor nem tudunk adatot másolni rá. A telefonunkat is használhatnánk adat hordozásra, azonban sokszor nincs nálunk adatkábel, amivel a géphez csatlakoztathatnánk. Ugyanakkor, ma már minden okostelefon rendelkezik wifivel, amivel létrehozhatunk kábel nélküli kapcsolatot egy számítógéppel. Mivel az okostelefon napjainkban mindenki számára egy nélkülözhetetlen eszköz, sokkal kevesebb esély van arra, hogy otthon felejtjük, mint egy pendrive-ot. Ezekből a feltevésekből kiindulva úgy gondolom, hogy hasznos lehet egy olyan program, amivel helyettesíteni tudjuk a pendrive-ot és annak funkcióit.

A HomeCloud-ot legkönnyebben úgy lehet leírni, mint egy olyan alkalmazást, amely lehetővé teszi különböző típusú fájlok vezeték nélküli átvitelét két eszköz között. Ez sok esetben hasznos lehet, ugyanis sok olyan ember van, aki nem igazán ért a számítógépek használatához, és nem ismer olyan programot, amivel ezt kivitelezni tudná. Alternatívaként használják például a Facebook üzenetküldés során való *fájlok csatolása* lehetőséget, vagy írnak egy e-mail-t saját maguknak, amiben csatolják a fájlokat, majd a másik eszközön fellépnek és letöltik. Azonban a fenti két opció nem mindig lehetséges, ugyanis mindkettő figyeli, hogy milyen típusú fájlokat töltünk fel, és túl nagy vagy gyanúsnak vélt fájlok esetén ellehetetleníthetik a fájlátvitelt. Az alkalmazásnak ilyen tekintetben egyszerűnek és könnyen kezelhetőnek kell lennie, aminek eleget is tesz, ugyanis lényegében két funkcióból áll: **Watch directory** azaz egy mappa figyelése, illetve **File transfer**, azaz fájl átvitel. A funkciók használatához szükség van arra, hogy az adott telefon, laptop, számítógép ugyanazon az alhálózaton, avagy Wifi hálózaton legyen elérhető. Annak érdekében, hogy az azonos hálózaton lévő eszközök kommunikálni tudjanak egymással, tudniuk kell egymás IP címét. Ehhez azonban először is szükség van a **Network discovery**-re, azaz a hálózat felderítésére, amely által tudomást szerezhetnek egymásról. A megvalósításához Multicast címzést

használunk, ami lényegében egy olyan fajta csoport kommunikáció, ahol az adott cím egy csoport címét reprezentálja, ahova több gép beléphet. Így, ha az egyik gép üzenetet küld a csoportba, úgy a csoportban lévő összes gép megkapja az adott üzenetet. A Multicast címzéshez a kivitelezésben a szállítói rétegnek a **User Datagram Protocol**-t (UDP) használjuk. A fájlátvitelnél egy fontos szempont, hogy amennyiben lehetséges, kerüljük el a csomagvesztést, továbbá a kapcsolat, ami létrejön a két eszköz között az megbízható és biztonságos legyen. Ebből kiindulva a tényleges fájlátvitelhez a szállítói rétegnek a **Transmission Control Protocol**-t (TCP) használjuk. A működés során az egyik fél a szerver, a másik pedig a kliens (File Transfer), de mindkét fél futtathat egyszerre több szervert és klienst is (Watch directory).

A projekt elkészítése során sok nehézség és váratlan esemény ütötte fel a fejét a probléma bonyolultsága, a probléma megoldása során használt elemek korlátjai, illetve egyéb tényezők miatt. A felmerült problémákat és alkotó elemeket a dolgozat későbbi részeiben részletezni fogom. Kezdésként a programot felhasználói szemszögből mutatom be.

2. Felhasználói dokumentáció – HomeCloud

2.1. A megoldandó feladat

A szakdolgozat célja egy olyan multiplatformos (jelenleg Windows és Android) alkalmazás létrehozása, amelyet a felhasználó könnyedén használhat, és pár kattintással lehetővé teszi, hogy az egyik eszközön levő fájlok a másik eszközre kerüljenek, mindezt kábeles összeköttetés nélkül. Ehhez képesnek kell lennünk az adott alhálózaton lévő eszközök felderítésére. A pendrive kiváltásához a programnak képesnek kell lennie a számítógépen levő fájlokat az Androidos okostelefonra másolni, majd onnan egy másik számítógépre küldeni.

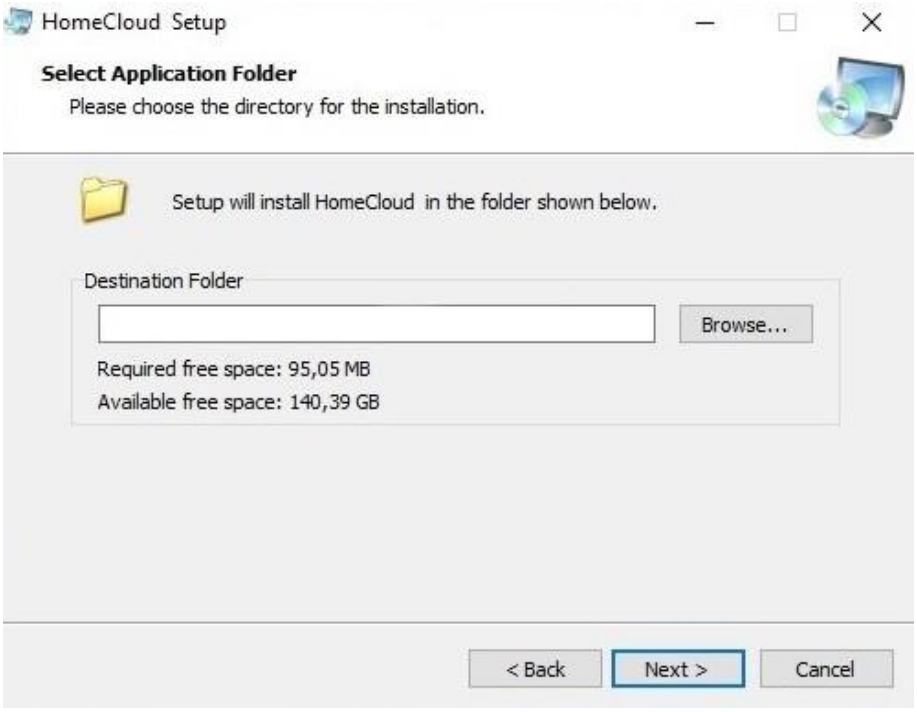
2.2. Rendszerkövetelmények

- Operációs rendszerek:
  - Microsoft Windows XP, Vista, 7, 8, 8.1, 10 – 32 vagy 64 bites kiadás
- RAM:
  - A program átlagos körülmények között 70-150 MB rendszermemóriát használ, tehát ennek függvényében a program használatakor a fent említett memóriának mindenképpen elérhetőnek kell lennie. A Microsoft Windows verziók különböző rendszermemória igénnyel rendelkeznek, továbbá az egyéb akkor futó programok memória használatának figyelembe vételével, az adott számítógépnek legalább 2-4 GB RAM-mal szükséges rendelkeznie.
- Háttértár:
  - A telepítő 50,72 MB tárterületet foglal, a program pedig 95,05 MB -ot.

2.3. Telepítés

A Microsoft Windows bármelyik verziójánál a telepítés ugyanúgy zajlik: az egér bal gombjának két kattintásával a **Setup.exe** fájlon egy felugró ablakban megjelenik a telepítő. A telepítés 6 lépésből áll, a lépések között a **Back** gombra kattintva visszakerülünk az előző oldalra, a **Next** gombra kattintva pedig eljutunk a következő lépéshez. Az első oldal a telepítő üdvözlő lapja. A második oldalon (lásd 1. ábra) kiválaszthatjuk az útvonalat, ahova a programot telepíteni szeretnénk. Az elérési utat kiválaszthatjuk egy útvonal fehér téglalapba való bemásolásával vagy a **Browse** gombra

való kattintással. Ez alatt láthatjuk, hogy a program telepítéséhez mennyi szabad helyre van szükség, illetve mennyi áll rendelkezésre. A harmadik oldalon megadhatjuk, hogy a programról készüljön-e egy parancsikon a Start Menüben vagy az Asztalon, illetve megváltoztathatjuk a program csoport nevét amennyiben szeretnénk. A negyedik oldal egy visszatekintés, mely által ellenőrizhetjük az eddig kiválasztott opciókat. Az ötödik oldal a telepítés állapotát mutatja, mely végeztével átkerülünk az utolsó, hatodik oldalra, ami a telepítés végeztéről és sikerességéről informál bennünket. A **Finish** gombra kattintva bezárul a telepítő, és amennyiben bejelöltük a **parancsikon készítése a Start Menüben vagy az Asztalon** menüpontot, látnunk kell, hogy a parancsikon sikeresen létrejött. A telepített csomag a következő fájlokat tartalmazza: Uninstall.exe és Uninstall.ini (a program eltávolításához), HomeCloud.ico (parancsikon kép), HomeCloud.jar (a fejlesztői környezetből kinyert projekt). A jar fájl futtatását a HomeCloud.bat fájl teszi lehetővé. Mivel a projekt egy JavaFx projekt, és JavaFx elemeket használ, a projekt számára lehetővé kellett tenni ezek elérését, így a **javafx-sdk-16** mappában a JavaFx SDK 16-os verziója található. Végezetül a mappa tartalmaz még egy HomeCloud.vbs fájlt, az elkészült parancsikon erre a fájlra hivatkozik. A fájl a HomeCloud.bat fájlt indítja el a parancssori ablak felugrása nélkül, így a parancsikon elindításával csak a program ablaka jelenik meg.



1. ábra A telepítő második oldala

2.4. Eltávolítás

A program eltávolítására 3 lehetőségünk is van. Az egyik módszer a telepítés során megadott útvonalban, a **HomeCloud** mappában található, ahol az **Uninstall.exe** fájl kiválasztásával el tudjuk távolítani a programot. Ekkor megjelenik egy ablak, ami megkérdezi, hogy biztosan törölni akarjuk-e a programot, majd a **Yes** gombra kattintva elkezdődik az eltávolítás. A folyamat végeztéről egy felugró ablak informál minket, ezt az **Ok** gombra kattintva zárhatjuk be. A másik lehetőség az, hogy a Vezérlőpultban a **Program eltávolítása** során megjelent listában jobb klikkel a **HomeCloud**-ra kattintva az **Eltávolítás** opciót választjuk. További lehetőség az, ha a **Setup.exe** telepítőre kattintunk, ez ugyanis telepítés előtt ellenőrzi, hogy a program egy korábbi verziója már telepítve volt-e. Ekkor ugyanis a telepítő meghívja az uninstaller-t, amiben két felugró ablak kérdezi meg, hogy biztosan el szeretnénk távolítani-e a programot.

2.5. Az első indítás előtti előkészületek

Az első elindítás előtt meg kell győződnünk arról, hogy a **Java Runtime Environment** (JRE) vagy **Java Development Kit** (JDK) a számítógépen már telepítve van. Ezt könnyen megtehetjük például, ha a parancssorba (CMD) beírjuk a **java -version** parancsot, ez ugyanis megmutatja a telepített verziót, ellenkező esetben nem fogja felismerni a parancsot, vagy hibát fog jelezni. A program a JDK 16-os verziójával készült el, így érdemes minél frissebb verziót használni/telepíteni.

A következő lépésben a Windows tűzfal beállításait kell szemügyre vennünk. Ezt Microsoft Windows 10-en például megtehetjük a *Vezérlőpult > Rendszer és biztonság > Windows Defender tűzfal* helyen, ahol a baloldalon megjelenő **Speciális beállítások** -ra kattintva előugrik a szükséges ablak. Ha szeretnénk meggyőződni arról, hogy az alkalmazás használata során, az eszközeink kellőképpen látják egymást, akkor a következőképpen tesztelhetjük: az egyik számítógépen a parancssorba írjuk be az **ipconfig** parancsot, nézzük meg mit ír az **IPv4 Address** címnél, majd a másik számítógépen ugyancsak a parancssorba írjuk be a **ping** parancs után, az előző eszközön az IPv4 Address címnél látható számot. Ekkor láthatjuk, hogy ha a ping csomagok szállítódnak, akkor minden rendben van. Androidon is elvégezhetjük ezt a műveletet, amennyiben letöltünk egy olyan alkalmazást, ami lehetővé teszi a pingelést. Ha nem sikerül a másik eszköz pingelése, abban az esetben a **Bejövő szabályok**-nál

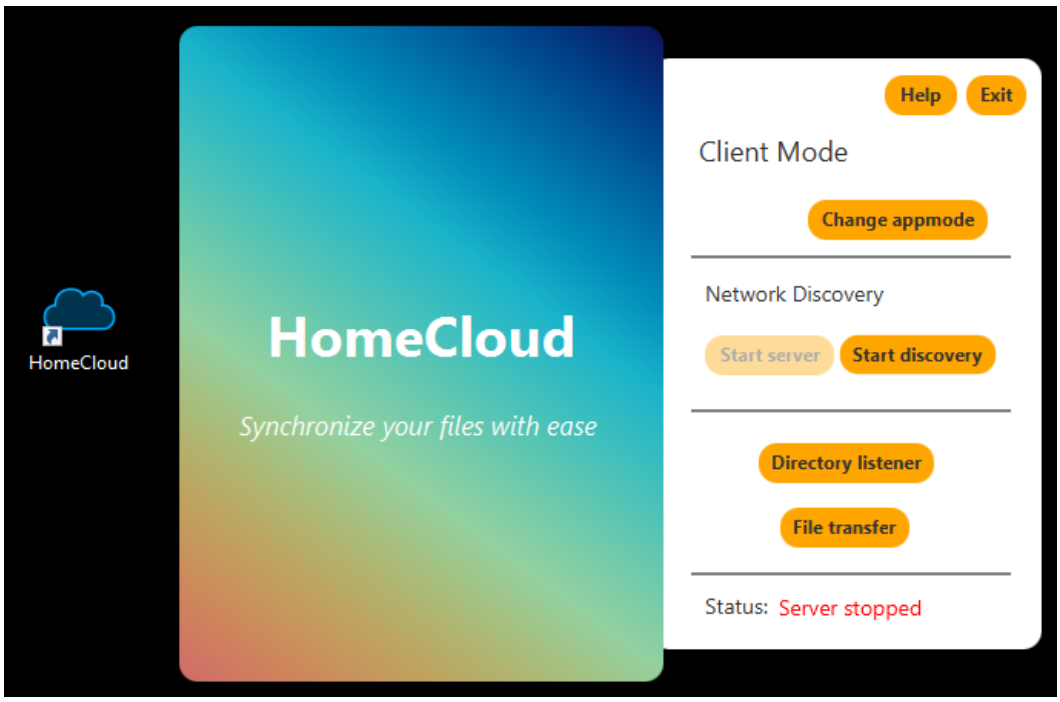
keressük meg a **Fájl- és nyomtatómegosztás (Echo kérés – ICMPv4-be)** szabályokat és nézzük meg, hogy aktiválva vannak-e már, mert ha nincsenek, akkor aktiválnunk kell őket. Ezt engedélyezve, a pingelésnek működnie kell, így meggyőződhetünk arról, hogy az eszközök látják egymást.

Ahhoz, hogy a számítógép kommunikálni tudjon az Android telefonokkal, de néhány esetben még ahhoz is, hogy más számítógéppel kommunikáljon, szükség van a Windows tűzfal egyes portjainak megnyitására. Ezt úgy tehetjük meg, hogy a bal oldalon lévő **Bejövő szabályok**-ra kattintva a jobb oldalon kiválasztjuk az **Új szabály**-t, majd a felugró ablakban kiválasztjuk a **Port** típusú szabályt. Kattintsunk az **UDP** opcióra, majd az **Adott helyi port**-hoz írjuk be 4446 és 4447 portszámot, ugyanis az UDP Network Discovery ezeket a portokat használja. Ezután a következő oldalon az **Engedélyezze a kapcsolatot**, majd az azt követőn a **Mikor lép érvénybe ez a szabály?** kérdésnél az összeset bejelölhetjük. Végezetül, hogy később emlékezzünk erre a szabályra adjunk neki egy rá jellemző nevet, például HomeCloudUDP. Ismételjük meg az előbbi folyamatot, hozzunk létre egy új szabályt, de most TCP-t. Az **Adott helyi port**-hoz írjuk be a 4448,4449,4450,4460-es portszámokat, és hasonlóan adjuk neki például a HomeCloudTCP nevet. Ha a fentebb említett feltételeket teljesítettük, akkor immár biztosak lehetünk abban, hogy a Program sikeres működésre kész.

2.6. Az alkalmazás használata futás közben

Az alkalmazás elindításához a parancsikonra való második kattintás után megjelenik a főmenü (lásd 2. ábra). A modern stílus miatt, az alkalmazásnak nincs fejléce, ezáltal nem lehet a fejléc általános funkcióit használni, mint az ablak mozgatás, lekicsinyítés, felnagyítás, bezárás és átméretezés. Az előbb felsoroltak egyikére a következő alternatívák születtek: az ablakok mozgathatóak, ehhez az ablakban az egeret valahol le kell nyomnunk, majd a bal gomb lenyomva tartásával és az egér mozgatásával az ablak a kívánt pozícióba helyezhető. A minimalizálás megoldható az alkalmazás Tálcában látható ikonjára kattintásával. A program bezárható egyik lehetőségként ugyanitt, ha a Tálcán található ikonra visszük az egeret, majd a Tálca felett felugró kis ablak jobb felső sarkában lévő **X** -re kattintunk. További lehetőség erre a főmenü jobb felső sarkában lévő **Exit** gomb, ugyanis ezzel is bezárható a program. Az Exit gomb mellett egy **Help** nevezetű gombot láthatunk, erre kattintva a

felugró ablakban az alkalmazás használatáról, illetve funkcióiról olvashatunk röviden. A felugró Help ablakból, az **Ok** gombbal tudunk kilépni.



2. ábra Az alkalmazás főképernyője. (Jelenlegi kép: Network discovery utáni állapot)

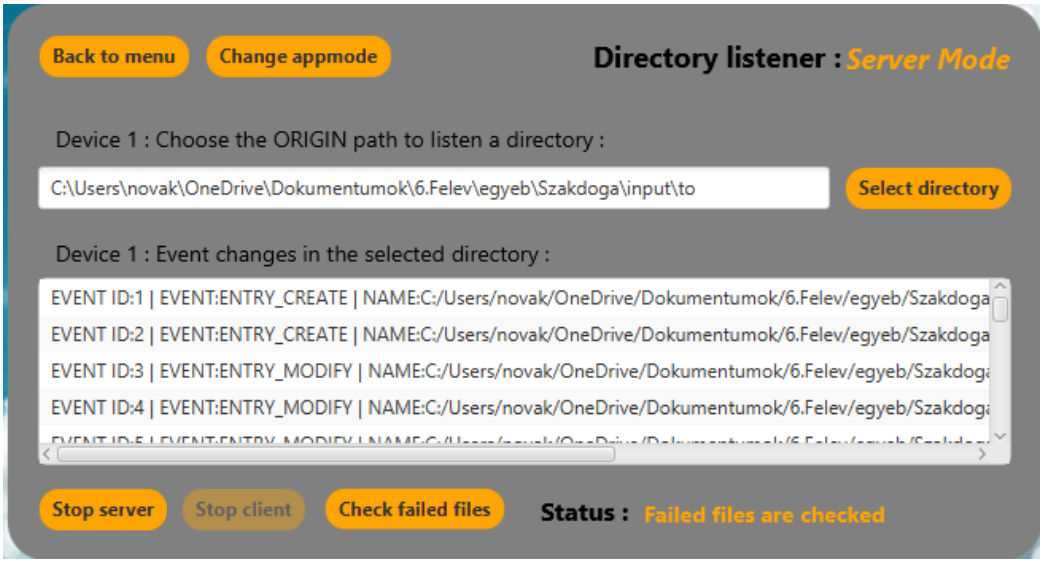
Az alkalmazást két eszközön kell egyidőben használni, azonban csupán egy eszközön való használata is lehetséges, habár így nem tudja kifejteni az eredeti kívánt hatást. Annak érdekében, hogy egyszerűbb legyen megkülönböztetni, hogy melyik eszközön pontosan mit kell csinálni, a Help gomb alatt található szöveg, az **appmode** ad tájékoztatást. Két mód lehetséges: **Server Mode** és **Client Mode**. Az állapotok között a **Change appmode** gombra kattintva tudunk változtatni. A változtatás látható eredményeképpen, a Network Discovery alatt Server Mode-ban a **Start server** gomb érhető el, míg Client Mode-ban való váltáskor a gomb letiltódik, ezzel együtt a Start **discovery** gomb aktiválódik. Az alkalmazás funkcióinak használatához először a **Network Discovery** szükséges, ugyanis ezzel a módszerrel jegyzi meg a két eszköz a másik fél IP címét. A Network Discovery során csupán annyi a teendőnk, hogy az egyik eszközön Server Mode-ban - ami az alap mód - rákattintunk a Start server gombra, és ezzel elindítjuk az UDP szervert. A gombnyomásra a Start server gomb felirata megváltozik Stop server gombra, ezzel lehetőséget adva a szerver megállítására, ha azt szeretnénk. Eközben a másik eszközön átváltunk a Change appmode gomb megnyomásával a Client Mode-ba, majd megnyomjuk a Start discovery gombot, amivel elküldünk egy csomagot a kijelölt csoportba. A gomb megnyomására a Start discovery

gomb felirata megváltozik Stop discovery gombra, ezzel lehetőséget adva az aktuális gépen lévő szerver megállítására, ha azt szeretnénk. Amennyiben a két eszköz ugyanazon az alhálózaton szerepel, és ha elvégeztük az alkalmazás első indítása előtti előkészületeket, a következőket tapasztaljuk: pár másodperc alatt a Client Mode-ban lévő eszközön automatikusan a Stop discovery gomb felirata visszaáll az elindításhoz szükséges Start discovery feliratra. A Server Mode-ban lévő eszközön a Stop server gomb felirata visszaáll az elindításhoz szükséges Start server feliratra. Ez azt jelenti, hogy a két eszköz felfedezte egymást, elmentették egymás IP címét. Ezáltal mindkét eszközön elérhetővé válik a **Directory listener** és **File transfer** funkció (az Androidon levő verzióban a Directory listener funkció Watch a directory néven szerepel). A főmenü alján egy **Status** feliratú szöveg található, ami mellett piros színnel láthatjuk informáló és értesítő jelleggel, hogy mi történik jelenleg az alkalmazásban, mint például, hogy a szerver elindult vagy megállt.

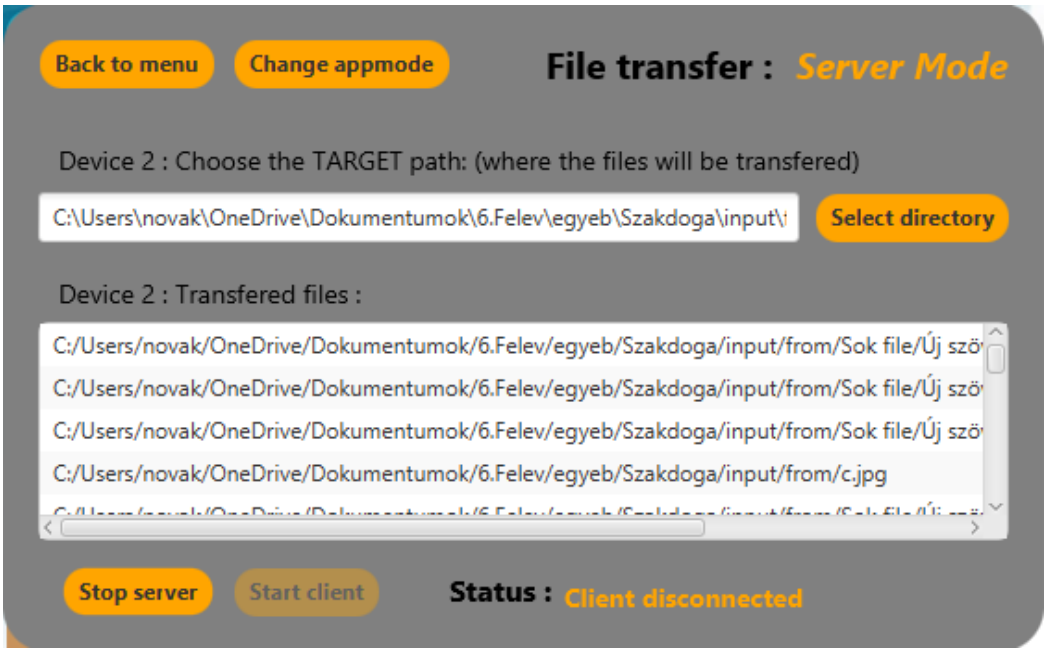
A Directory listener (lásd 3. ábra) vagy File transfer (lásd 4. ábra) kiválasztása után felugrik egy ablak, ahol ugyanúgy le van tiltva a fejléc, emiatt ezen funkciók használata hasonlóan működik a főmenühöz. Amennyiben mozgatni szeretnénk az ablakot, ugyanúgy az ablakon belül kell kattintanunk, majd lenyomva tartva az egéren a bal gombot, az ablakot oda helyezzük, ahova szeretnénk. A két funkció felugró ablakának kinézete hasonló, az egyedüli különbség az ablakokban található leíró szöveg, illetve, hogy a Directory listener egy plusz gombbal rendelkezik. Emiatt, hogy ne tévesszük össze mikor melyik lehetőséget használjuk, a felugró ablak jobb felső sarkában lévő szöveg megmutatja, hogy melyik gombot választottuk a főmenüben. Továbbá, az ablak használata során az aktuális állapotról is itt szerezhetünk tudomást. Az ablakok megjelenése során a főmenü letiltásra kerül, így amennyiben a másik lehetőséget szeretnénk választani vagy ki szeretnénk lépni, a főmenüből tehetjük meg. Az ablakok bal felső sarkában két gombot láthatunk. Az első a **Back to menu,** aminek a megnyomására visszakerülünk a főmenübe. A második gomb a **Change appmode,** ami a főmenüben használt Change appmode gombbal megegyezően, az alkalmazás aktuális használati állapotát tudja megváltoztatni. Az ablakban található következő elem az adott elérési út kiválasztására szolgál. Erre az egyik lehetőség az, hogy bemásoljuk a szövegdobozba a megnyitott mappa elérési útját. A másik opcióval a szövegdoboz melletti **Select directory** nevű gombra kattintunk, ami előhozza felugró ablak

formájában a Windowsban használt fájlkezelő rendszert, ahol kiválaszthatjuk a kívánt mappát. A kiválasztás után a mappa elérési útvonala bemásolódik a szövegdobozba, így láthatjuk a teljes útvonalat. A kiválasztott útvonal a File Transfer és Directory listener funkciók során, mindkét módban más szereppel rendelkezik. A Directory listener Server Mode-ban való használata során azt a mappát és a mappában lévő almappákat jelöli, amit figyelni fogunk (Origin path). A Client Mode-ban az útvonal a másik eszközön lévő azon helyet reprezentálja, ahova a fájlokat tárolni szeretnénk (Target path). A File Transfer Server Mode-ja során ugyanazt a mappát jelöli az útvonal, ahova a fájlokat tárolni szeretnénk (Target path). Client Mode-ban azt a mappát jelzi, aminek tartalmát át szeretnénk küldeni (Origin path). Mindkét funkcióban a Start Server és Start Client gombokat az ablak legalsó sorában találhatjuk. A gombok használata megegyező a főmenüben levőkkel: a Start Server a Server Mode-ban, míg a Start Client a Client Mode-ban érhető el. Elindulás után a Start – felirat átvált Stop feliratra, amivel majd ismételt kattintásra a szerver és kliens megállása eredményezhető. Fontos, hogy a gombok csak akkor indíthatóak el, ha már kiválasztottuk az adott mappa elérési útvonalát. A Directory listener funkcióban a Start client mellett szerepel egy **Check failed files** gomb. Ezt a gombot az alkalmazás bezárása, avagy a főmenüre való visszatérés előtt kell használni. A gomb lehetővé teszi, hogy leellenőrizzük, hogy a behúzott mappában minden fájl átvitelre került-e. Megtörténhet ugyanis, hogy a mappa figyelése során az adott könyvtárban bekövetkező változások az azok által okozható anomáliák miatt nem kerülnek követésre. A gomb megnyomásakor összevetjük a már átküldött fájlokat a figyelt könyvtárba behúzott fájlok listájával, majd azok a fájlok, amelyek nem kerültek átvitelre újra elküldésre kerülnek. Ha a fájlok ellenőrzése és átküldése véget ért, akkor a Status üzenetnél egy **Failed files are checked** üzenetet láthatunk, ugyanis a Status itt is informáló jelleggel szerepel. A Start Server felett egy üres négyzet található. Ennek jelentősége mindkét funkcióban ugyanaz: az adott fájlok vagy események átküldésének megjelenítése. A File transfer funkció Server Mode-jában a megérkezett fájlok listája szerepel, míg Client Mode-ban az összes elérhető talált fájlt kilistázza, amit a megadott mappában találtunk. A Directory listener Server Mode-jában a mappában és almappáiban történt eseményekről értesülhetünk. Itt kétfajta eseményt tartunk számon: ENTRY CREATE – fájl létrehozása és ENTRY MODIFY – fájl módosítása. Egy fájl

behelyezése során, az adott fájl több eseményt is generálhat. Végezetül pedig Client Mode-ban a már átküldésre került fájlok / események jelennek meg.



3. ábra A Directory listener ablak



4. ábra A File transfer ablak

3. Felhasználói dokumentáció – HomeCloudMobile

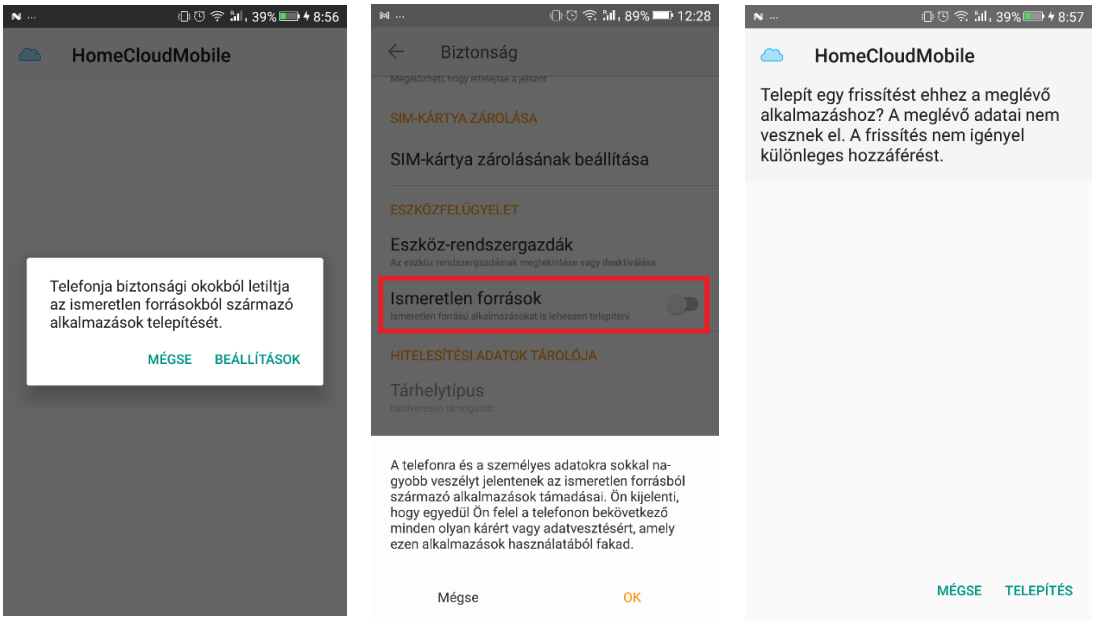
3.1. Rendszerkövetelmények

Az alkalmazás okostelefonon vagy tableten, nagyobb körben való használatához szükséges, hogy az adott készülék minimum 21-es API szinttel rendelkezzen (Android 5.0 – Lollipop nevezetű mobil operációs rendszer). Ezáltal a felhasználó egyedül a **File transfer** funkcióhoz férhet hozzá. A **Watch a directory** funkció használatához a készülék minimum 26-os API szinttel kell rendelkezzen (Android 8.0 – Oreo nevezetű mobil operációs rendszer).

Az alkalmazás felhasználói felülete úgy lett tervezve, hogy mindig az adott készülék kijelzőjéhez igazodjon, azonban a kényelmes használat érdekében legalább egy 5” -es kijelzővel rendelkező készülék ajánlott.

Az alkalmazást telepítő apk kiterjesztésű fájl megközelítőleg 2.5MB-ot, míg maga az alkalmazás körülbelül 7.5MB-ot foglal.

3.2. Telepítés

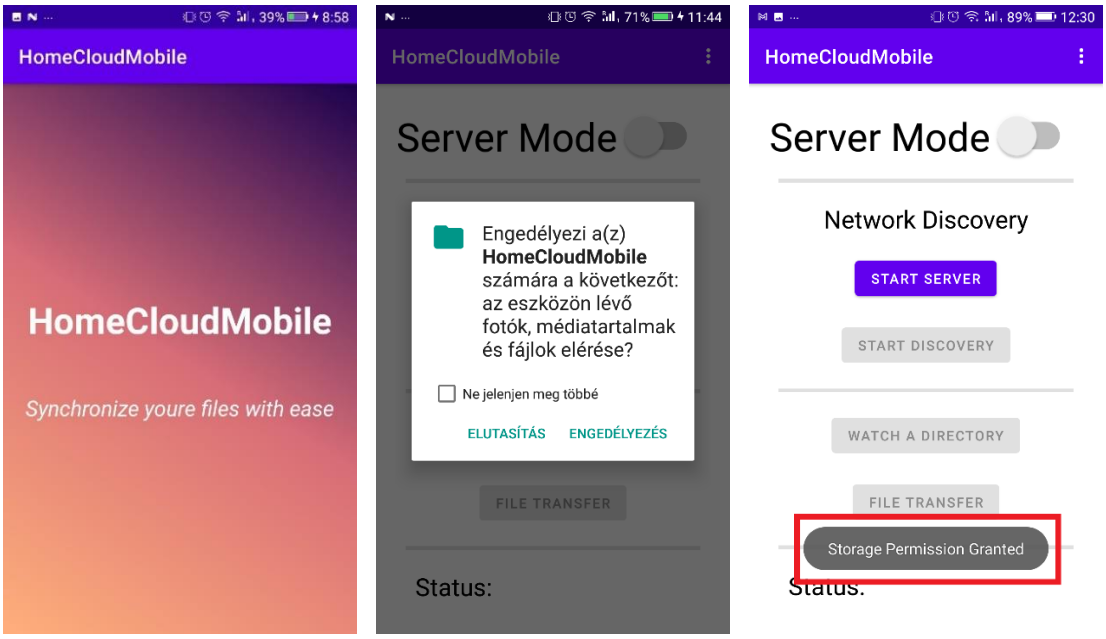


5. ábra Telepítési folyamat

Az alkalmazás telepítéséhez másoljuk a HomeCloudMobile.apk fájlt a készülékre. Megnyitásakor amennyiben nincs engedélyezve az alkalmazások ismeretlen forrásokból való telepítése, az 5. ábra első képén látható ablak fog elénk ugrani. Ekkor a beállításokban, a Biztonság menüpont alatt legalább a telepítés erejéig

engedélyeznünk kell az Ismeretlen forrásokat (lásd 5. ábra második képe, pirossal bekeretezett rész). Az alkalmazás semmilyen veszélyt nem jelent, így az elcsúsztatáskor megjelent felugró ablakban nyugodtan választhatjuk az **OK** gombot. Ezután visszatérhetünk a telepítőre (lásd 5. ábra harmadik képe), majd válasszuk a Telepítés lehetőséget, ezt követően az alkalmazás néhány másodperc alatt a készülékre telepítődik. Végezetül visszatérhetünk a Biztonság részhez, majd letilthatjuk az Ismeretlen forrásokat.

3.3. Az alkalmazás használata futás közben

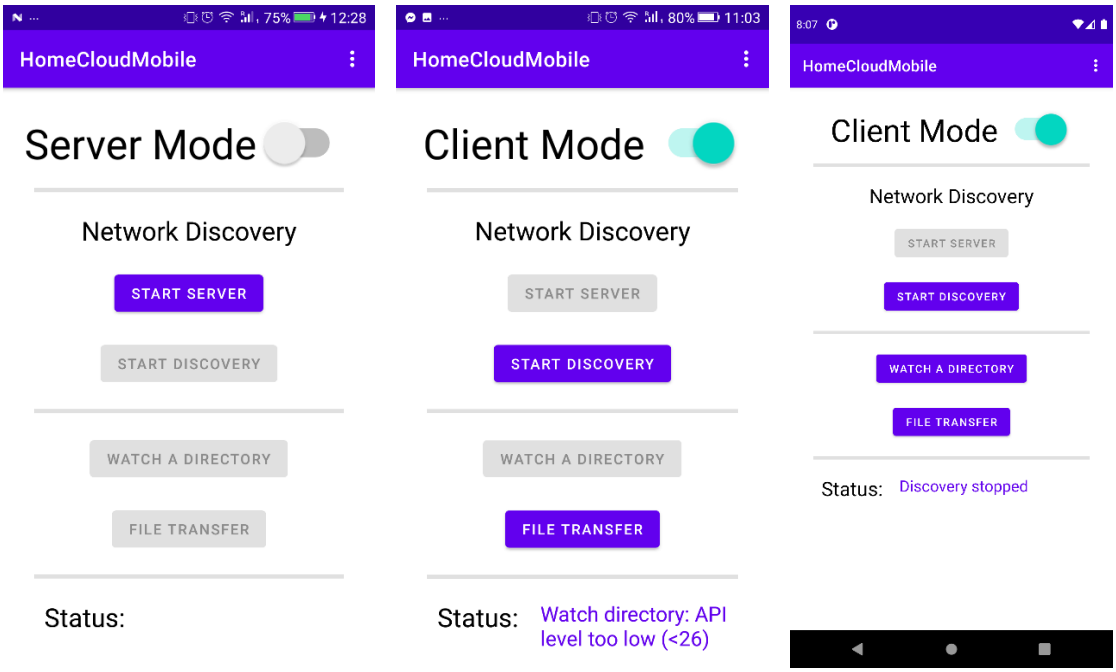


6. ábra Az alkalmazás megnyitása a használat előtt

Az alkalmazás mindenkori megnyitásakor egy üdvözlő lap fogad minket (lásd 6. ábra első képe), mely 4 másodperces várakozás után eltűnik, majd ezt követően megjelenik a főoldal. Az alkalmazásnak a helyes működéséhez hozzá kell férnie a telefon belső és külső tárhelyéhez a fájlkezelő használata, és a fájlok másolása miatt. Az engedély megadása érdekében egy, a 6. ábra második képéhez hasonló ablak fog megjelenni. Amennyiben elutasítjuk, az ablak bezárul és az alkalmazás azonnal kilép, továbbá minden egyes indítás során, ameddig nem engedélyezzük a tárhelyhez való hozzáférést, az ablak fel fog ugrani. Amennyiben engedélyeztük, egy **Storage Permission Granted** szöveg (lásd 6. ábra harmadik képe) fogja majd jelezni nekünk, hogy többet nem kell találkoznunk az engedélykérő ablakkal.

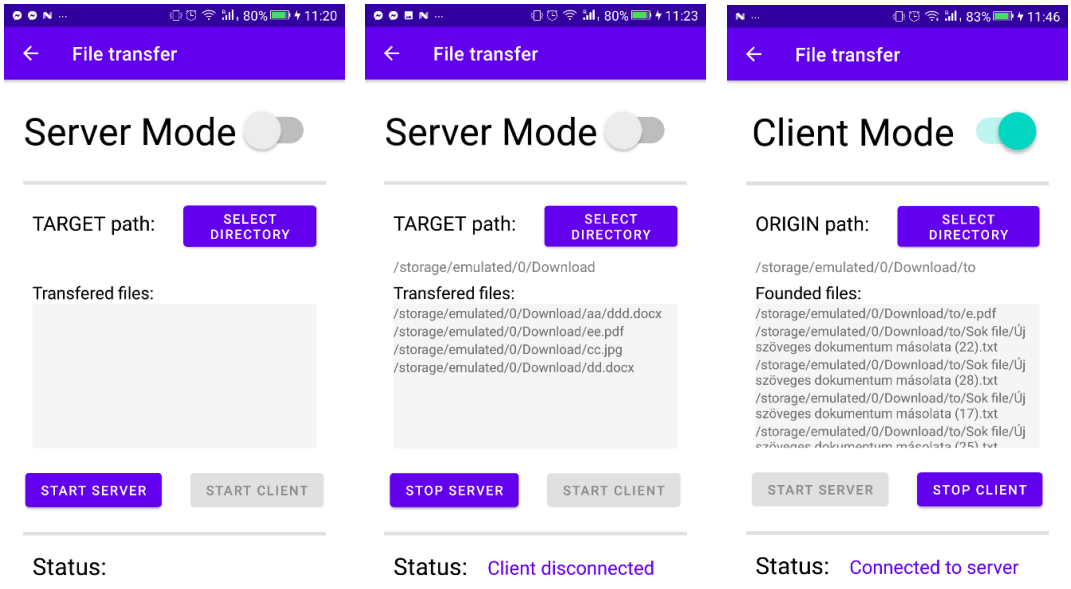
A főmenü használata és működése megegyező a számítógépen levővel, annak

ellenére, hogy egy-egy elem más funkcióval vagy kivitelezéssel került megoldásra. A **Help** gomb a főmenü jobb sarkából, Androidon átkerült az Overflow menübe. A funkcionalitása ugyanaz, kiválasztásakor részletes leírást és segítséget kaphatunk az alkalmazás használatához. A Help mellett ebben a legördülő menüben megtalálhatjuk az **Exit** gombot is, amit ha kiválasztunk, akkor az alkalmazás bezárul. Az alkalmazásból kiléphetünk a telefonunkban levő, középső Home nevezetű gombbal is. A főmenü legtetején, egy úgynevezett Switch helyettesíti a számítógépen levő Change appmode gombot. Az elv ugyanaz: ahhoz, hogy a Network Discovery működjön, az egyik eszközt Server Mode-ban kell használnunk, míg a másikat Client Mode-ban (A főmenü Network Discovery előtti állapotát lásd 7. ábra első képén). A Server Mode-ban levő eszközön, a Start Server gombbal indíthatjuk el az UDP szervert, Client Mode-ban pedig a Start Discovery gombbal kezdhetünk neki a futó szerverek felderítésének. Elindítás után mindkét gomb megállítható, ugyanis a gombokon lévő Start szöveg helyett a Stop veszi át a szerepet: a Start Server megnyomása után a gomb szövege átvált Stop Server-re, a Start Discovery gomb szövege pedig Stop Discovery-re. Amennyiben a két eszköz megtalálta egymást, úgy a gombok automatikusan visszaálnak a megállítható Stop feliratú állapotból az indítható Start feliratú állapotba. Annak nincs különösebb jelentősége, hogy melyik eszközön vagyunk Server vagy Client Mode-ban. Ha nem végeztük el a számítógépen levő előkészületeket használat elött, azt tapasztalhatjuk, hogy semmi nem történik, ugyanis a Windows tűzfal blokkolni fogja a bejövő kliensek csatlakozását. Ez bekövetkezhet, ha nem engedélyeztük az említett portokat, például, hogy ha a számítógépet Server Mode-ban használjuk és már elindítottuk a szervert, a telefonon pedig Client Mode-ban már rányomtunk a Start Discovery gombra. A Network Discovery végezetével, a készülék Android verziójától függően két opció lehetséges: ha a készüléken az Android verziója kisebb, mint 8.0, akkor csak a File transfer gomb lesz elérhető (lásd 7. ábra második képe), ha pedig legalább 8.0-ás, akkor mindkettő (lásd 7. ábra harmadik képe)



7. ábra A főmenü Network Discovery előtti, illetve az utáni állapotai

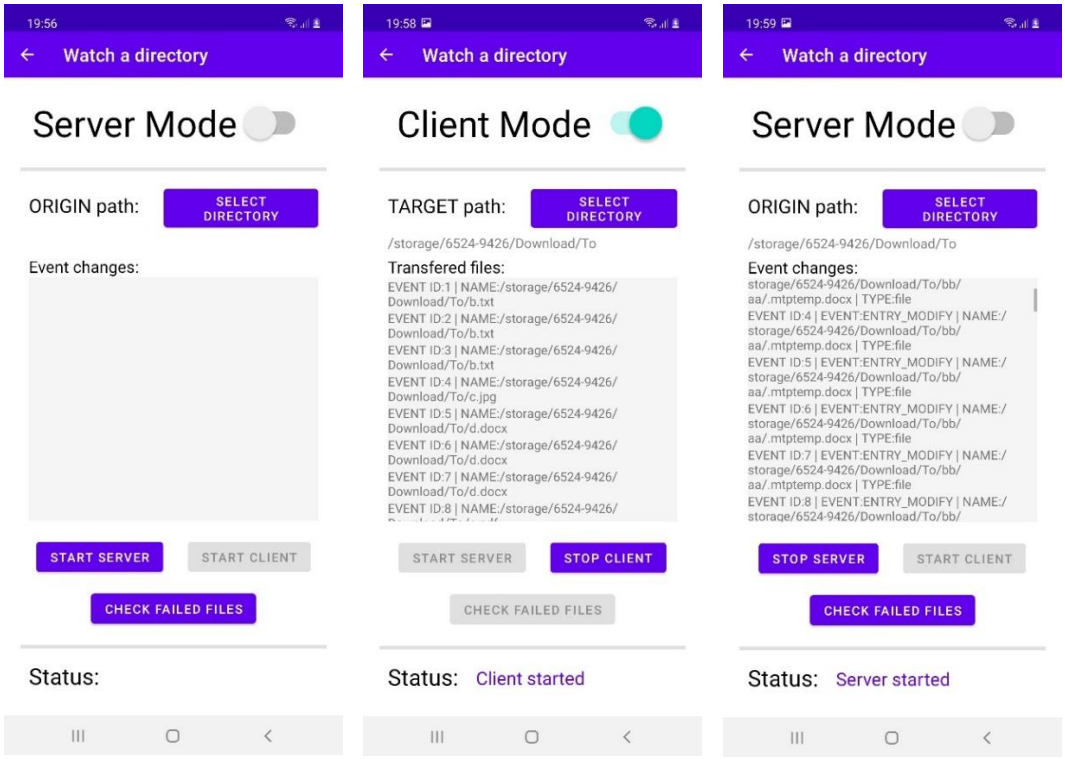
A File transfer funkció kiválasztása után a 8. ábra első képén látható oldal jelenik meg. Az ablak bal felső sarkában lévő visszafele irányuló nyíl megnyomásával visszakerülünk a főmenübe. Mivel az Android más koncepciót használ, a főmenüre való visszatérés során a főmenüt úgy kezeli, mintha frissen nyitottuk volna meg, így az újbóli használathoz ismét el kell végeznünk a Network Discovery -t.



8. ábra A File transfer funkció használata előtti és utáni állapotai

A File transfer használata során, először ki kell választanunk a **SELECT DIRECTORY** gombra kattintva az adott mappát, ahova a fájlokat tárolni, illetve ahonnan a fájlokat küldeni szeretnénk. Az alkalmazás Server Mode-ban való használatakor a TARGET path

azt az útvonalat jelenti, ahol a fájlokat majd tárolni fogjuk, míg a Client Mode-ban megjelenő ORIGIN path azt az útvonalat jelenti, ahonnan küldjük őket. A mappaválasztás után a felirat és gomb alatt megjelenik a kiválasztott útvonal. Ezek után már elindíthatjuk a szervert vagy a klienset. Server Mode-ban a **Transfered files** alatt láthatjuk majd a fájlokat, amik megérkeztek (lásd 8. ábra második képe), Client Mode-ban a **Founded files** alatt láthatjuk a fájlokat, amiket küldeni fogunk (lásd 8. ábra harmadik képe).



9. ábra A Watch a directory funkció használata előtti és utáni állapotai

A Watch a directory funkció kiválasztásakor a 9. ábra első képén látható ablak tárul elénk. Mint láthatjuk, kinézetében sokban hasonlít a File transfer funkcióhoz. A korábbihoz hasonlóan, először a **SELECT DIRECTORY** gomb által ki kell választanunk a megadott elérési utat, mely Server Mode-ban való használatakor az ORIGIN path, ami azt a könyvtárat jelöli, amelyet figyelni fogunk. A Client Mode-ban levő TARGET path azt az útvonalat jelöli, ahol a fájlokat majd tárolni fogjuk. Kiválasztás után az útvonal meg fog jelenni a felirat és gomb alatt. A Server Mode-ban való használat során, miután kijelöltük a mappa figyelését, kiléphetünk az alkalmazásból és áthelyezhetjük a fájlokat az adott mappába. Ha nem akarunk kilépni, egy másik lehetőségként a telefont fizikailag összeköthetjük egy számítógéppel, ezáltal a telefonon bejelölhetjük az adott mappa figyelését, majd a számítógépről bele húzhatjuk a szükséges fájlokat, és a

telefon majd elküldi őket a fogadó eszköznek. Végezetül, mivel lehetséges, hogy némely mappában levő fájl behúzását nem vette észre a rendszer, kilépés elött a **CHECK FAILED FILES** gomb megnyomásával tudjuk leellenőrizni a sikertelenül elküldött fájlokat. A behúzás során született események az **Event Changes** felirat alatt lesznek kilistázva (lásd 9. ábra második képe). Client Mode-ban való használatkor mi vagyunk a fogadó fél, így egy másik eszközön, a figyelt mappába behúzott fájlokat fogjuk megkapni. A megérkezett fájlok a **Transfered files** felirat alatt lesznek kilistázva (lásd 9. ábra harmadik képe).

Az alkalmazás telefonon való használatakor győződjünk meg arról, hogy a fájlok küldésekor és fogadásakor a két eszközön ne szerepeljen ugyan olyan névvel rendelkező fájl. Ez akkor szükséges, ha a telefonunk nem tudja kezelni a dátum módosításokat, mivel így a fájlok frissebb módosítási dátummal rendelkező szinkronizálása/régiek felülírása nem garantált. Az Android készülékeken való dátum módosítás kiszámíthatatlan, némely eszköz egyszerűen képtelen a fájlok dátumának módosítására. Van olyan eszköz, amely pontosan beállítja az összes dátumot, de az is előfordulhat, hogy a kapott fájlok egy részénél rendesen beállítódik a dátum, míg a másik részénél nem. Ekkor alapértelmezettnek az átküldés időpontját állítja be a rendszer utolsó módosítási dátumként. Amennyiben nem akarunk szinkronizálni, vagy nem fontos, hogy mikor volt a fájl utoljára módosítva, például csak szállítani szeretnénk, akkor a problémával nem kell foglalkoznunk.

3.4. Státusz üzenetek

Az alkalmazás működése során a következő státusz üzenetekkel találkozhatunk úgy az asztali alkalmazásban, mind az Androidosban:

- **„Server started”**
  - Főmenü esetében jelzi, hogy elindult a Network Discovery során használt UDP szerver
  - File transfer funkció esetében jelzi, hogy elindítottuk a szervert, most már készen állunk a fájlok fogadására
  - Watch a directory funkció esetében jelzi, hogy elindult a szerver, azaz most már figyeljük az adott mappában történő változásokat
- **„Server stopped”**

- Főmenü esetében jelzi, hogy megállítottuk, vagy megállt magától a Network Discovery során használt UDP szerver
- File transfer funkció esetében jelzi, hogy bezártuk a szervert, nem fogadhatunk több fájlt
- Watch a directory funkció esetében jelzi, hogy abbahagytuk az adott mappában történő változások figyelését.

➢ **„Client connected”**

- Főmenü esetében jelzi, hogy a Network Discovery során használt UDP szerverhez csatlakozott a másik eszköz.
- File transfer funkció esetében jelzi, hogy csatlakozott a szerverhez a kliens, kezdődik a fájl átvitel

➢ **„Changed to "Client Mode" ”**

- Bármelyik oldalon vagy funkcióban legyünk, ez a felirat azt jelzi, hogy Server Mode-ból Client Mode-ba váltottunk

➢ **„Changed to "Server Mode" ”**

- Előzőhöz hasonló, jelzi, hogy Client Mode-ból Server Mode-ba váltottunk

➢ **„Connection established”**

- A főmenüben, a Server Mode-ban levő eszközön jelenik meg, tudatva, hogy a két eszköz felfedezte egymást, és most már használható a két funkció

➢ **„Server discovered”**

- A főmenüben, a Client Mode-ban levő eszközön jelenik meg, tudatva, hogy a két eszköz felfedezte egymást, és most már használható a két funkció

➢ **„Watch directory: API level too low (<26)”**

- Csak Androidon jelenik meg, a főmenüben, Network Discovery után abban az esetben, ha a készülék Android verziója kisebb mint 8.0

➢ **„Discovery started”**

- A főmenüben jelenik meg, jelzi, hogy elindult a hálózat felderítése

➢ **„Discovery stopped”**

- A főmenüben jelenik meg, jelzi, hogy megállítottuk vagy leállt magától a hálózat felderítése

➢ **„Directory was chosen successfully”**

- File transfer és a Watch a directory funkció esetében jelzi, hogy sikeresen

kiválasztottuk a megadott elérési útvonalat

- **„Choosing of the directory failed.”**
  - File transfer és a Watch a directory funkció esetében jelzi, hogy nem sikerült kiválasztanunk a megadott elérési útvonalat
- **„Client started”**
  - File transfer funkció esetében jelzi, hogy a Client Mode-ban levő eszköz nekikezdett a fájlok átküldésének
  - Watch a directory funkció esetében jelzi, hogy készen állunk fogadni az eseményeket/fájlokat
- **„Client stopped”**
  - File transfer funkció esetében jelzi, hogy a Client Mode-ban levő eszköz befejezte a fájlok átküldését, vagy pedig mi állítottuk meg az átküldést
  - Watch a directory funkció esetében jelzi, hogy abbahagytuk az események fogadását
- **„Client disconnected”**
  - File transfer funkció esetében jelzi, hogy lecsatlakozott a kliens, befejeződött/abbamaradt a fájlok átküldése
- **„Connected to server”**
  - File transfer funkció esetében jelzi, hogy a Client Mode-ban levő eszköz csatlakozott a szerverhez
- **„Checking failed files”**
  - Watch a directory funkció esetében jelzi, hogy elindult a sikertelen fájlok ellenőrzése
- **„Failed files are checked”**
  - Watch a directory funkció esetében jelzi, hogy befejeződött a sikertelen fájlok ellenőrzése és átküldése

4. Fejlesztői dokumentáció

4.1. Fejlesztőeszközök

Az alkalmazás Java nyelvben íródott, ami egy erősen típusos objektum-orientált nyelv. A választás indokai között szerepelt első sorban a nyelv platform függetlensége, ugyanis a Java nyelvben íródott programok bármilyen típusú számítógépen és operációs rendszeren elfutnak, amennyiben egy JRE már telepítve van. Személyes preferenciám alapján, mivel a Microsoft Windows típusú operációs rendszereket kedvelem, emiatt a telepítő jelenleg csak Windows-on elérhető, azonban nagyon kevés erőfeszítéssel elérhető lehet például macOS-en vagy Linuxon. A nyelv felhasználható Android-ra való portolás során is, habár az Android más compile-er és package-elési metódusokat használ, az úgynevezett **Android Application Package**-t (APK), ennek ellenére a java forráskód szépen lefut. A nyelv széles körben népszerű, folyamatosan fejlődik, rengeteg használható könyvtárat és egy elérhető dokumentációt tartalmaz, további fontos szempontként támogatja a szál- és hálózat kezelést és rengeteg online fórumon lehet keresni egy adott problémára való választ, mint például a StackOverflow.

A projekt fejlesztési környezeteként az IntelliJ IDEA Community Edition 2021.1 verzióját használtam. A választás azért esett rá, mivel könnyű a használata, gyorsan hozzá lehet szokni, ezáltal a kódolási részt sokkal könnyedebbé, természetesebbé és kielégítőbbé teszi, továbbá a legokosabb IDE a világon, analizálja a kódot, jó kódolási segédlete van, és okos hiba analízissel rendelkezik.

Az alkalmazás **Graphical User Interface**-e (GUI) a JavaFx szoftver platform használatával valósult meg. A választás azért esett erre, mert a JavaFx ki kínál egy a Scene Builder nevezetű alkalmazást, amivel nagyon könnyen és gyorsan, drag and drop módon elkészíthető egy olyan GUI, ami komponensekben gazdag, előrehaladott, modern érzéssel és kinézettel tölt el. Így tehát a Scene Builder 16-os verzióját használtam. A JavaFx és a Scene Builder használatának másik előnye, hogy a kódot könnyebben olvashatóvá teszi, azáltal, hogy a **Model–View–Controller** (MVC) programtervezési mintát használja. Ez azt jelenti, hogy egy adott ablakért lényegében 3 fájl felel: a Model a Scene Builder-rel elkészített .fxml kiterjesztésű fájl, ami lényegében az oldal tartalmát képviseli. A View betölti és megjeleníti ezt az adott Model-t egy

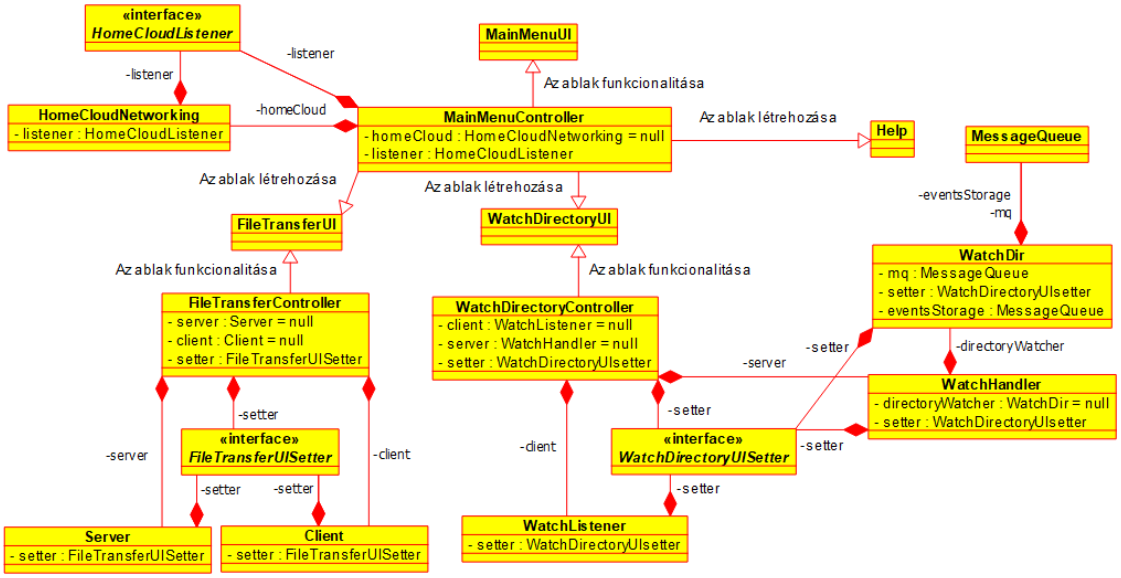
ablakban, ami majd a felhasználói felületté válik. Végezetül pedig a Controller felel majd az ablakban lévő gombok, elemek funkcionalitásának kifejtéséhez. A Java és az FXML kód jól használható együttesen, ehhez mindössze annyit kell tennünk, hogy a Scene Builder-ben amikor a .fxml fájlban ID-t, vagy onClick függvénynevet állítunk be, a Controller osztályban egyszerűen hivatkozhatunk az adott elemre a @FXML annotációval.

4.2. A fejlesztés előkészületei

A fejlesztés megkezdésének első lépése a JDK legfrissebb verziójának letöltésével és telepítésével kezdődött. Ezt követően, szükség volt a JAVA_HOME környezeti változó beállítására, ami a telepítés helyén lévő bin mappát kapta meg útvonalaként. Következőkben az IntelliJ, a JavaFx SDK és Scene Builder letöltése következett. Végezetül, ezek összehangolására volt szükség: az IntelliJ-ben a telepitett JDK és SDK útvonalának beállítása, a **Libraries**-ban a **javafx sdk lib** mappájának beállítása, végezetül a **Run Configurations**-ban, a **VM options**-ban a következő sort kellett bemásolni: „**--module-path "C:\ProgramFiles\Java\javafx-sdk-16\lib" --addmodules = javafx.controls,javafx.fxml**”. Ezen beállítások után az elkészített alap projekt már futtatható is.

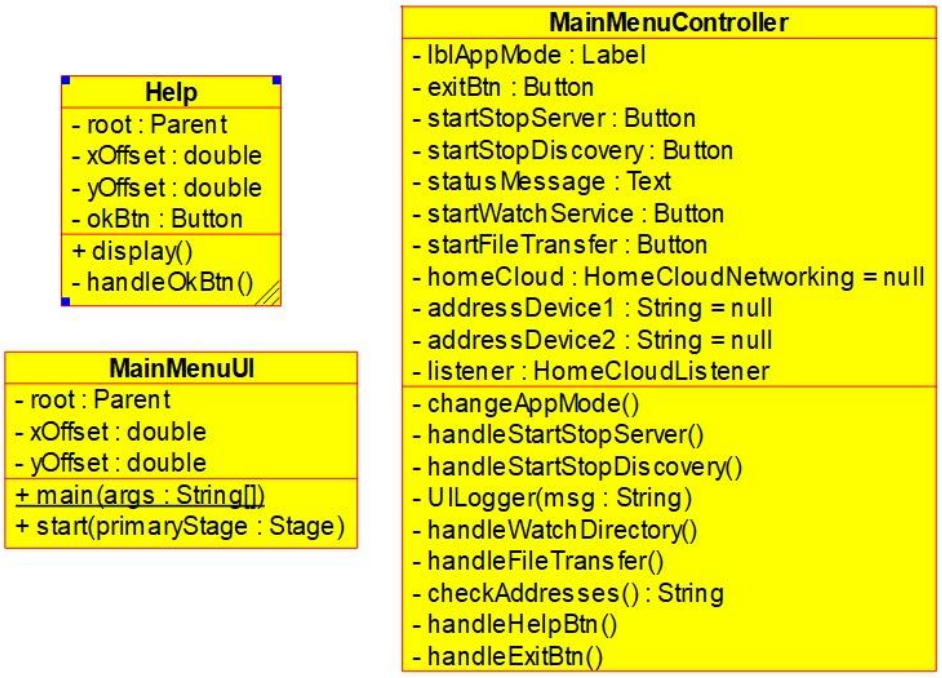
4.3. Az alkalmazás osztályai

A 10. ábrán az alkalmazás UML osztály diagramja látható. A továbbiakban ezeket az osztályokat, illetve a felhasználásukat fogom bemutatni.



10. ábra Az alkalmazás osztályainak UML osztálydiagramja

4.4. A főmenü



11. ábra A Help, MainMenuUI, MainMenuController osztályok

A főmenü elkészítése [1] során az MainMenu.fxml fájl a Scene Builder-ben, drag

and drop módszerrel gyorsan elkészíthető volt. Lényegében két AnchorPane-ből tevődik össze, amelyek mögötti AnchorPanek színét transparent, azaz átlátszóra állítottam. A bal oldali rész díszelemként szolgál, háttérszínét egy három színből álló color-gradient határozza meg. A modern hatás érdekében a menüben lévő elemeket, például ablakok széle, gombok, TextField-ek során 1em széles border-radius lett beállítva. A jobb oldali, fehér hátterű lapon helyezkednek el a program használatához szükséges gombok.

A kódban három fájl felel a főmenüért. Az első a MainMenu.fxml, ami a felhasználó elött lévő felület elemeit tartalmazza. A MainMenuUI osztály (lásd 11. ábra) fő szerepe a .fxml fájl betöltése és itt található az alkalmazás futtatásához kellő main metódus. Miután betöltöttük, letiltjuk az alapértelmezett alkalmazás fejlécet, pontosabban áttetszővé tesszük, beállítunk egy ikont, ami az alkalmazás ikon, illetve a Tálcán lévő alkalmazás kis ablakának ikonja. Mivel letiltottuk az alkalmazás fejlécet, így végezetül ennek megoldásához felül kell definiálnunk az ablakban történő egér lenyomását, és lenyomva tartását.

4.4.1. A MainMenuController osztály

A MainMenuController osztály (lásd 11. ábra) szerepel a főmenü funkcionalitásáért. Az Exit gomb működéséért a handleExitBtn metódus felel. Lényegében bezárjuk az alkalmazást, de még azelőtt megnézzük, hogy a Network Discovery során használt szerver es kliens létezik és fut-e, mert amennyiben igen, megállítjuk és lezárjuk őket, ugyanis, ha nem kezelnénk ezt le, a következő futásnál nem lehetne elindítani a szervert, mert az adott port cím még foglalt lenne. A handleWatchDirectory, handleFileTransfer, handleHelpBtn metódusok a nevükből adódóan a Directory listener, File transfer és Help ablakok megjelenítéséért felelnek azáltal, hogy létrehoznak egy példányt a WatchDirectoryUI, FileTransferUI és Help osztályokból, majd meghívják a display metódusokat az osztályoknak az ablakok megjelenítéséért.

Annak érdekében, hogy a felhasználó által kiváltott interakciók hatására az alkalmazás felületén változás történjen futás közben, szükségünk van a Platform.runLater hívásra, ez ugyanis a JavaFx alkalmazás szálján, egy meghatározatlan jövőbeli időben fogja futtatni az adott Runnable-t. Ennek az alkalmazására például a

UILogger-ben van szükség, amikor a Status feliratú szövegnél lévő üzenetet meg szeretnénk változtatni a képernyőn. Annak érdekében, hogy egy időben a szálak ne blokkolják egymást miközben a statusMessage változót változtatják az UILogger metódus hívására, a **synchronized(statusMessage)** sort használjuk, ugyanis ezzel csak a statusMessage változóra tiltjuk meg az egyazon időben történő változtatást.

A Start server és Start discovery gomb működéséért a handleStartStopServer és handleStartStopDiscovery metódus felel, melyek majdnem azonosak. Mindkét esetben azt figyeljük, hogy milyen szöveg szerepel az adott gombon. Ha a gombon a **Start server/discovery** szerepel, akkor először is megnézzük, hogy hoztunk-e már létre példányt a HomeCloudNetworking osztályból, mert ha nem, akkor először példányosítanunk kell. Ha pedig már létrehoztunk, akkor csak rendre meghívjuk a HomeCloudNetworking startServer és startDiscovery metódusát, majd ahhoz, hogy ha már elindítottuk ne tudjuk még egyszer elindítani, csak megállítani, a gombokon lévő **Start server/discovery** szöveget lecseréljük a **Stop server/discovery** szövegre. Így a következő alkalommal, amikor megnyomjuk a gombot azt figyeljük, hogy a szövegben a Start szó helyett Stop van, ekkor ugyanis meghívjuk a HomeCloudNetworking stopServer és stopDiscovery metódusát, ami kiváltja majd a megállást. Végezetül pedig a gombokat letiltjuk, hogy addig, amíg teljesen le nem állt ne lehessen megint megnyomni a Stop gombokat.

A **Change appmode** gomb működéséért a changeAppMode metódus felel és az alkalmazás szerver/kliens módban való használatát teszi lehetővé. Erre azért van szükség, mivel 2 eszköz használata esetén egyik szerver, másik kliens szerepet fog betölteni. Lényegében a már kiírt szöveget figyeljük, hogy milyen módban vagyunk, így ha például az adott szövegben a **Server Mode** -ban van, akkor tudjuk, hogy **Client Mode** -ba kell átrakni, és ez alapján aktiváljuk/deaktiváljuk a start-stop gombokat.

Mivel a futás folyamán sokszor változtatjuk az ablakon lévő elemeket más osztályokból például gombok szövegét, státusz üzenetet, annak elkerüléseként, hogy a felhasználói felületért felelős Controller osztályt kelljen mindig átadni, interfészeket hoztam létre az utólagos módosítás érdekében. A főmenü módosítására a HomeCloudNetworking osztály által a HomeCloudListener interfész szolgál. A Controller-ben példányosítjuk az interfészt, majd, amikor a handleStartStop-Server/Discovery metódusokban példányosítjuk a HomeCloudNetworking osztályt,

paraméterül megkapja ezt az interfészt. Az HomeCloudListener egyes metódusai informáló jellegűek, mint például a serverStarted, discoveryStarted és clientConnected, amiket akkor hívunk meg ha elindult az UDP szerver vagy kliens, illetve, ha a klienstől jövő üzenetet megkapta a szerver. A serverStopped és discoveryStopped metódusok majd akkor kerülnek meghívásra, ha a szervert és klienset kezelő szálak befejezték működésüket. Csak ekkor tesszük újra elérhetővé a **Stop server/discovery** gombokat, majd a Stop szót lecseréljük a Start-ra ezzel is jelezve, hogy most már újból elindíthatóak. A connectionEstablished és discoveredServer metódusokat akkor hívjuk meg amikor már véget ért a Network Discovery, mivel ezzel kialakult a kapcsolat és így elmentjük a másik eszköz IP címét. A címeket a Controller osztályban tároljuk el, és miután megtettük, elérhetővé tesszük a Directory listener és File transfer gombokat deaktivált állapotukból. Az eltárolt IP címeket a Directory listener és File transfer ablakokat megvalósító osztályok paraméterül kapják meg, aminek ellenőrzésére a checkAddresses metódus szolgál.

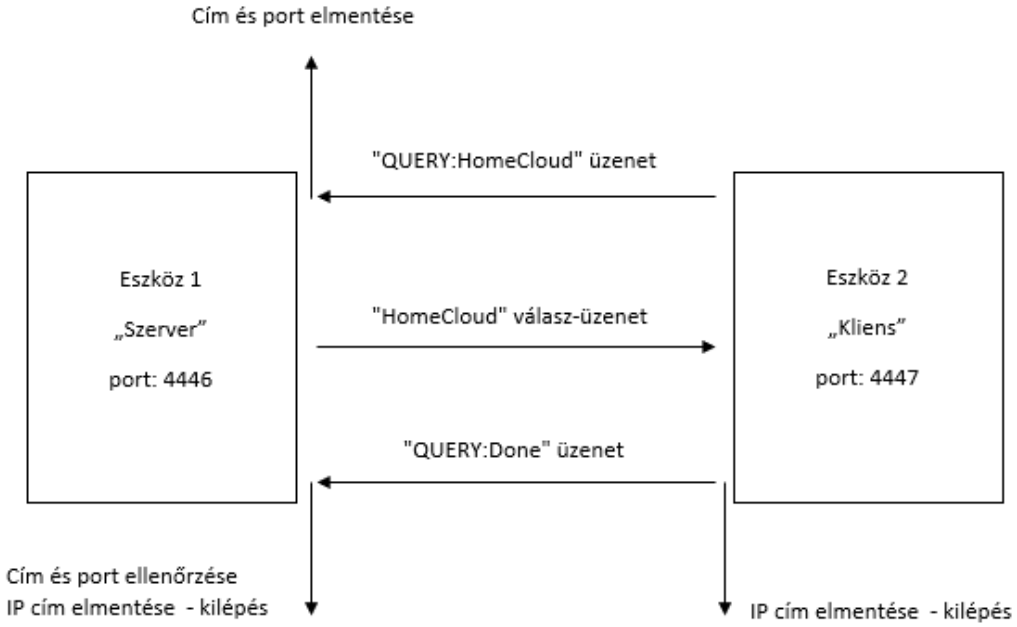
4.5. Help ablak

A Help ablak megjelenítéséhez két fájl, a Help.fxml és Help.java szükséges. Az fxml fájl létrehozása ugyan úgy a Scene Builder-ben történt. Az ablakban csak az alkalmazás használatát segítő szöveg, illetve egy **OK** gomb található. A Help.java (lásd 11. ábra) fájl a MainMenuUI-hoz hasonló: a Help ablak .fxml fájljának betöltéséért felel. Letiltjuk az alapértelmezett alkalmazás fejlécet, beállítjuk a tálcán megjelenő ablak ikonját és hasonlóan felül kell definiáljuk az ablakban történő egér lenyomását, és lenyomva tartását, végezetül pedig amíg az ablak aktív, addig lehetetlené tesszük a főmenü elérését. Az OK gomb működését a handleOkBtn metódus végzi, amiben csak bezárjuk az ablakot és ezzel visszatérünk a főmenübe.

4.6. Network Discovery

A Network Discovery során az UDP Multicast címzést használjuk. Az UDP és TCP a legismertebb szállítási protokollok. Az UDP-t rövid, gyors üzenetek küldésére szokás használni, amikor fontosabb, hogy gyorsan küldjünk el valamit, minthogy megbízhatóan. Ahogy a nevében is benne van, datagram alapú, tehát a csomag megérkezésére nincs semmilyen garancia, ugyanis nem nyugtázható. A Multicast címzés lehetővé teszi, hogy egy adott IP címre több eszköz felcsatlakozzon, így, ha

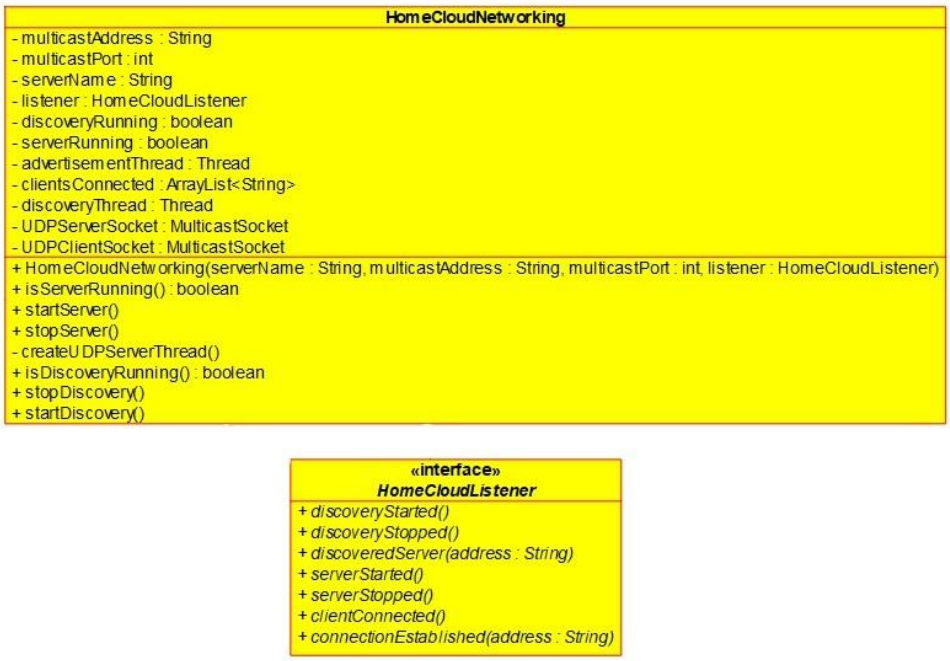
valaki üzenetet küld, akkor az szétszóródik a csoportban. Ez tökéletes az alhálózat felfedezésére amikor nem tudjuk a másik fél IP címét, ugyanis csak egy üzenetet kell küldenünk a csoportba, és ha azt a másik fél megkapja, onnantól tudni fogja az IP címünket, majd, ha ő válasz üzenetet küld, úgy mi is tudni fogjuk az övét. Hogy megértsük az alhálózat felderítése esetünkben miként valósul meg, tekintsük meg a 12. ábrán látható kommunikációs diagram-ot amely a szerver és kliens közötti üzenetcserét mutatja be.



12. ábra A Network Discovery kommunikációs diagramja

4.6.1. A HomeCloudNetworking osztály

A Network Discovery folyamatot, mely lehetővé teszi a másik eszköz felderítését és IP címének az elmentését, a **HomeCloudNetworking** osztály (lásd 13. ábra) valósítja meg. Konstruktorában paraméterül kapja meg a Multicast csoport nevét, címét és portját, illetve a felhasználói felület módosítására szolgáló HomeCloudListener interfész osztálynak egy példányát (lásd 13. ábra). Az osztály szerkezetileg két részre osztható: szerver és kliens részre.



13. ábra A HomeCloudNetworking osztály és HomeCloudListener interfész

A szerver részén az **isServerRunning** metódussal kérdezzük le a főmenüben kilépés elött, hogy fut-e még a szerver. A szerver indításáért a startServer metódus felel, melyben először is azt vizsgáljuk, hogy a szervert létrehozó szál létezik-e, mert ha már létezik, akkor return-olunk, hogy ne lehessen kétszer elindítani. Meghívjuk a **createUDPServerThread** metódust, igazra állítjuk a futásért felelős boolean változót, majd végezetül a felhasználói felületen feltüntetjük, hogy elindult a szerver. A szerver megállítását a stopServer metódus végzi: ha nem létezik még a szerver futásáért felelős szál, akkor return-olunk, hogy ne lehessen megállítani mikor nem is fut a szerver. Mivel a MulticastSocket receive metódusa blokkoló jellegű, emiatt ennek feloldásához szükség van a MulticastSocket.close hívásra, ami lezárja a szervert, ezzel feloldva a blokkolást.

A **createUDPServerThread** metódus a szerver működéséért felel, melyben egy advertisementThread nevű szálat hozunk létre. Ebben először létrehozzuk a MulticastSocket-et a multicastPort-on, ami az esetünkben a 4446-os port, majd belépünk az adott „224.0.0.1” IP című csoportba, létrehozzuk az in-output csatornákat (DatagramPacket-eket és a hozzájuk szükséges byte típusú buffereket), illetve egy ArrayList típusú listát, amiben a már csatlakozott klienseket fogjuk számon tartani. Egy while ciklust futtatunk addig amíg a futást igazoló serverRunning változó értéke igaz, továbbá a receive metódussal várunk/figyelünk, hogy jött e üzenet. Ha igen, akkor

megjelenítjük a felhasználói felületen, hogy kliens csatlakozott. Első sorban megvizsgáljuk, hogy a kapott üzenet **QUERY:serverName** típusú. Ha igen, akkor válaszolunk, és amennyiben a már csatlakozott kliensek között még nem szerepel az aktuálisan csatlakozott kliens, elmentjük az IP címét és port számát az adott listába. Erre azért van szükség, mert a kapcsolatot csak akkor nyilvánítjuk kialakítottnak, ha a már csatlakozott kliensek között szerepel az aktuális kliens, mert ezzel már tudjuk, hogy korábban már megkaptuk tőle a QUERY előtaggal ellátott üzenetet. Ezáltal az előbbire küldött válasz üzenetünkre most egy **QUERY:Done** típusú üzenetet kell kapnunk, ugyanis így a kapcsolatot létrejöttnek tekinthetjük és el tudjuk menteni a másik fél IP címét, amit követően megállíthatjuk a Network Discovery-t.

A kliens részén az **isDiscoveryRunning** metódussal kérdezzük le a főmenüben kilépés elött, hogy fut-e még a kliens. A megállításért a **stopDiscovery** metódus felel, melyben, ha nem létezik még a futásáért felelős szál, akkor return-olunk (hogy ne lehessen megállítani mikor nem is fut). Szerverhez hasonlóan itt is szükségünk van a MulticastSocket.close hívásra, ami lezárja a MulticastSocket-et ezzel feloldva a blokkolást.

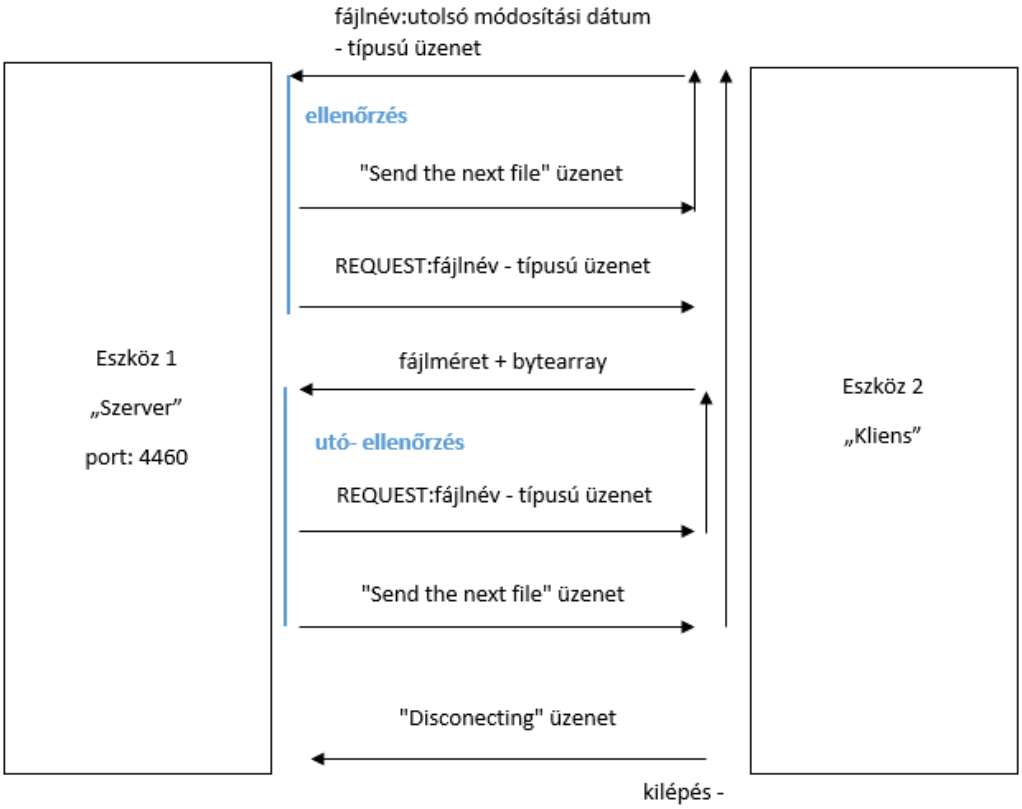
A kliens indításáért a **startDiscovery** metódus felel, melyben ugyan úgy azt vizsgáljuk, hogy a klienst létrehozó discoveryThread szál létezik-e, mert ha már létezik, akkor return-olunk, hogy ne lehessen kétszer elindítani. Azután létrehozzuk a szerver létrehozásáért felelős előbb említett szálat, igazra állítjuk a futásért felelős boolean változót, majd végezetül a felhasználói felületen feltüntetjük, hogy elindult a kliens. A szerver részhez hasonlóan először létrehozzuk az MulticastSocket-et a multicastPort + 1-en, ami így a 4447-es port lesz, majd belépünk az adott „224.0.0.1” IP című csoportba, létrehozzuk az in-output csatornákat (DatagramPacket-eket és a hozzájuk szükséges byte típusú buffereket). Két egymásban levő while ciklust futtatunk ugyan azzal a futás értékét igazoló discoveryRunning változóval. Az első ciklusban szétküldjük a **QUERY:serverName** típusú üzenetet a csoportban, majd a következő ciklusban várjuk a válasz üzenetet. Amikor megjött, akkor visszaküldjük a **QUERY:Done** üzenetet majd lementjük a másik eszköz IP címét, és megállítjuk a Network Discovery-t.

Habár az eddigiekben a könnyebb megkülönböztetés véget az egyik eszközt szervernek, a másik eszközt kliensnek említettem, valójában a két eszköz egyenlő félként működik olyan értelemben, hogy mindketten létrehoznak egy MulticastSocket-

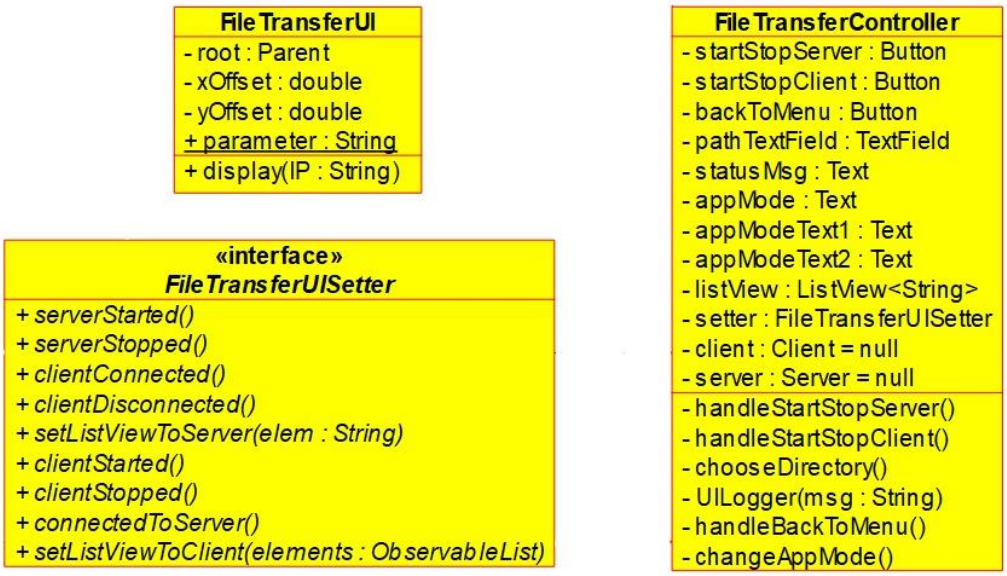
et, a szervernek nevezett eszköz a 4446-os porton, a kliensnek nevezett eszköz a 4447-es porton. Ha az üzenet váltásokat tekintjük, ahhoz, hogy eljussunk az IP cím eltárolásához, a szerver üzenetet fogad-küld-fogad, a kliens pedig küld-fogad-küld.

4.7. File transfer

A File transfer funkció biztosítja a fájlok átküldését és fogadását két eszköz között. Mivel a két eszköz már tisztában van a másik IP címével, ezért a fájl átvitel során így már használhatjuk a megbízható TCP protokollt. Annak érdekében, hogy könnyebben megértsük a két eszköz közötti fájl átvitel hogyan valósul meg, a 14. ábra bemutatja a szerver és a kliens közötti üzenetváltásokat.



14. ábra A File transfer kommunikációs diagramja



15. ábra A FileTransferUI és FileTransferController osztályok és FileTransferUISetter interfész

A File transfer funkció 6 részből áll: FileTransferUI, FileTransferController, FileTransfer.fxml, FileTransferUISetter, Server, Client. Ahogy az eddigi .fxml fájlok, úgy a FileTransfer.fxml fájl is a Scene Builder segítségével került összeépítésre. A FileTransferUI (lásd 15. ábra) tartalma hasonló a Help és MainMenuUI osztályokhoz: letiltjuk a főmenüt, fejlécet, ikont állítunk, betöltjük az fxml fájlt, felüldefiniáljuk az egér műveleteit az ablak mozgatásához. Mivel a File transfer során szükség van a másik eszköz IP címére, az ablakot előhozó display metódus paraméterben megkapja a főmenü MainMenuController osztályában eltárolt címeket. Ahhoz, hogy az itteni Controller osztályban is használni tudjuk a címet, a display metódus első sorában a kapott címet értékül adjuk a statikus String típusú parameter nevezetű változónak. FileTransferUISetter osztály (lásd 15. ábra) az ablak Server és Client osztályokból való módosításáért felel futás közben.

4.7.1. A FileTransferController osztály

A FileTransferController osztály (lásd 15. ábra) szolgál a felhasználói felület funkcionalitásáért. Néhány metódus azonos a MainMenuController-ben levővel, mint például az **UILogger**, ami a státusz üzenet frissítésére szolgál. A **handleBackToMenu** metódus az ablak bezárásáért és a főmenübe való visszatérésért felel. Bezárás elött megnézzük, hogy a példányosított client és server létezik és fut-e, mert ha igen, akkor bezárjuk őket. A **changeAppMode** metódus is hasonló célt szolgál, ezzel állítjuk az

ablak állapotát. A szöveget nézzük, hogy Server/Client Mode-ban vagyunk-e és ez alapján aktiváljuk/deaktiváljuk a Start Server/Client gombokat, változtatjuk a feliratukat, valamint az ablakban levő leíró szövegeket, illetve minden állapot váltás után leüresítjük ListView-t (az ablakban levő négyzet, ami a talált/átküldött fájlok megjelenítésére szolgál) azáltal, hogy az elemeinek beállítjuk a null-ot.

A **chooseDirectory** [4] metódus az adott útvonal kiválasztására való abban az esetben, amikor a **Select directory** gomb megnyomására a felugró ablakból a számítógép könyvtár szerkezetéből akarjuk kiválasztani az utat (másik lehetőség amikor már másoljuk az útvonalat és beillesszük az erre fenntartott TextField mezőbe). A megoldás megvalósításához a JFileChooser-t használjuk. A metódus első sorában beállítjuk, hogy a JFileChooser az operációs rendszerben a megszokott mappakiválasztó felülethez hasonló legyen. Beállítjuk a felugró ablak címét és hogy csak mappákat jelenítsen meg, ugyanis csak adott mappák útvonalára vagyunk kíváncsiak, továbbá, hogy mentse el a kiválasztott elem útvonalát a **Save**-re kattintva. Amennyiben sikeresen kiválasztottunk egy adott mappát, megpróbáljuk beállítani a TextField-nek a mappa útvonalát, ugyanis a Server és a Client majd innen szerzi meg a kiválasztott értéket. Végezetül megjelenítjük a felhasználói felületen a művelet siker/sikertelenségét.

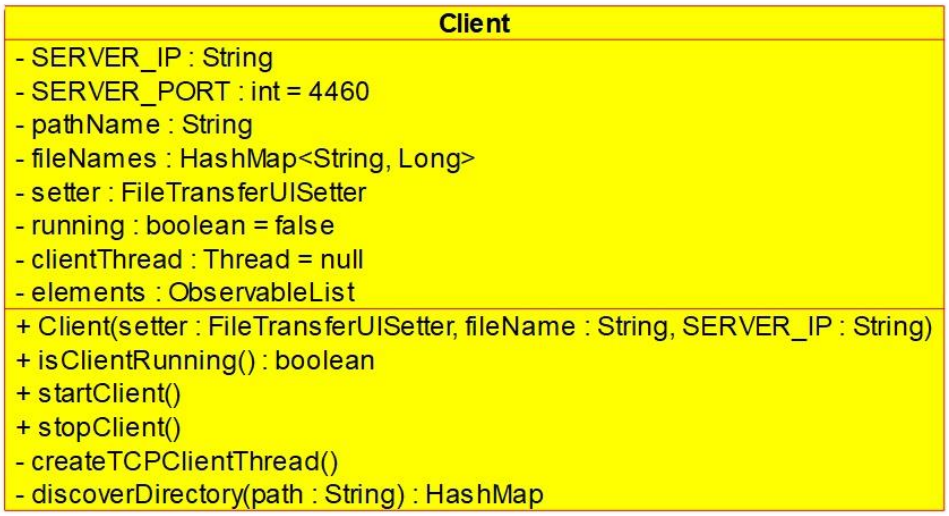
A **handleStartStopServer** és **handleStartStopClient** a **Start server** és **Start client** gombok elindításáért és megállításáért felel. Az elv ugyan az, mint a főmenüben használt Start/Stop gomboké. Figyeljük a gomb szövegét: ha a Start Server/Client szerepel, akkor utána megvizsgáljuk, hogy a TextField, amiben az elérési útvonal került kiválasztás vagy bemásolás útján nem üres-e, ugyanis csak ekkor lehet elindítani a Server vagy Client-et. Ha nem üres, akkor a benne levő String értékre először is egy replace-t hívunk, így az útvonalban esetlegesen szereplő „\” karaktereket kicseréljük a „/” karakterekre, ugyanis ezáltal megelőzzük String műveletek során, a speciális karakterek miatt történő potenciális problémákat. A továbbiakban, ha nem létezik még példány a Server-ből, akkor létre hozzuk. A Server eltárolja a legutoljára megadott elérési útvonalat, ezért, ha egy újabbat vél felfedezni, akkor az újabbat állítja be és ott fogja a fájlokat majd eltárolni. Végezetül elindítjuk a Servert, és a Start feliratot átírjuk Stop-ra. A Client esetében annyiban más a helyzet, hogy a String replace után minden alkalommal új Client példányt hozunk létre, amit ezt követően el is indítunk, és a Start

szöveget átírjuk Stop-ra. Amikor a gombon a Stop felirat található, mindkét esetben ugyan azt jelenti, hogy következő alkalommal kattintás után megállíthatjuk a Server/Client futását és ezzel deaktiváljuk a gombot.

4.7.2. A FileTransferUISetter interfész a FileTransferController osztályban

A File transfer ablak futás közbeni módosítására a Server/Client osztályokból a FileTransferUISetter interfész szolgál. A Controllerben példányosítjuk az interfészt, majd, amikor a handleStartStop-Server/Client metódusokban példányosítjuk a Server és Client osztályokat, paraméterül átadjuk nekik ezt az interfészt. A FileTransferUISetter egyes metódusai informáló jellegűek, mint például a serverStarted clientConnected, clientDisconnected, clientStarted, connectedToServer. Ezeket rendre akkor hívjuk meg, ha elindult a Server osztály szervere, ehhez a kliens csatlakozott vagy lecsatlakozott, elindult a Client osztály és csatlakozni próbál a szerverhez, majd végezetül, ha sikerült csatlakozni. A serverStopped és clientStopped metódusok majd akkor kerülnek meghívásra, ha a Server-t és Client-et kezelő szálak befejezték működésüket, ugyanis csak ekkor tesszük újra elérhetővé a Stop Server/Client gombokat. Végezetül a Stop szót lecseréljük a Start-ra, ezzel is jelezve, hogy most már újból elindíthatóak. Az ablakban van egy ListView típusú négyzet, ez Server esetén a már megérkezett fájlok, míg a Client esetében az adott mappában található összes küldhető fájl megjelenítésére szolgál. Ennek megvalósítása érdekében a setListViewToServer és setListViewToClient szolgál. A Client esetében az utóbbit használjuk, ami csak szimplán beállítja a listView változónknak az ObservableList típusú listát, ami a fellelhető fájlokat tartalmazza. Server esetében mivel a setListViewToServer metódust minden alkalommal meghívjuk, amikor egy fájl átérkezett, és ennek érdekében, hogy ez szinkronizált módon történjen, a metódus elején a **synchronized(listView)**-t használjuk. Mivel a fájlok a paraméterben csak egyesével szerepelnek, ellenőriznünk kell, hogy a listView inicializálva volt-e, mert ha nem, akkor az első elemet adjuk neki értékként. Amennyiben meg már létre volt hozva, minden alkalommal lekérjük a listát és hozzáadunk egy elemet.

4.7.3. A Client osztály



16. ábra A Client osztály

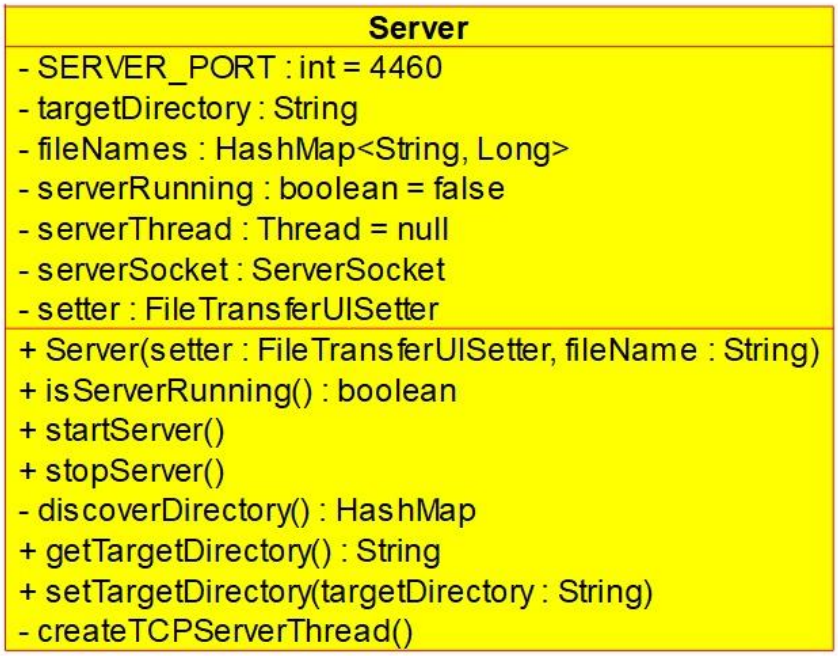
A Client osztály (lásd 16. ábra) konstruktorában megkapja a FileTransferUISetter példányát, az adott útvonalat, valamint a másik eszköz címét. Az isClientRunning-al a főmenüre való visszatérés elött ellenőrizzük, hogy a szál még fut-e, és ha igen megállítjuk. A startClient és stopClient metódusok nagyban hasonlítanak a HomeCloudNetworking szerver részében megtalálható startServer és stopServer metódusokhoz. Amiben különbség van, azok a változó nevek, továbbá, hogy a Client-ben nincsenek blokkoló metódusok, így a stopClient-ben nincs szükségünk a Socket.close hívásra.

A discoverDirectory metódussal történik az adott elérési úttal rendelkező mappa bejárása, amit a path paraméterben kapunk meg. A Files.walk [2] metódussal megpróbáljuk bejárni az adott könyvtár struktúra fa szerkezetének összes levelét, majd szűrűnk a rendes, reguláris fájlokra, amiket String típusúvá alakítunk és összegyűjtjük őket, ezáltal megkapjuk eredménynek a létező fájlokat. Bejárjuk az eredmény listát, és egy HashMap-ban eltároljuk a fájlok neveit, illetve utolsó módosítási dátumukat ügyelve arra, hogy a replace metódussal kicseréljük a „\” karaktereket „/” karakterekre. Végezetül összegyűjtjük a fájlokat az elements listába a felhasználói felületben való megjelenítésért. Ha nem történt hiba a művelet során, akkor visszatérünk a fentebb említett Hashmap-el, ellenkező esetben pedig null-al.

A createTCPClientThread [3] metódus a kliens oldali kommunikációért felel. Először csatlakozunk a szerverhez, amit a felhasználói felületen is megjelenítünk. Létrehozzuk a

szükséges in- es output csatornákat (Data-Input/Output-Stream), majd a fileNames-nek odaadjuk discoverDirectory metódus során összegyűjtött fájl neveket és módosítási dátumokat, és az így talált fájlokat megjelenítjük a felhasználói felületen is. Végig iterálunk a fileNames-en, és ezzel rákérdezünk a szervernél az adott fájlra. Várunk a szerver válaszára: ha a szerver válaszolt, és a **Send the next file** üzenet jött, akkor rákérdezünk a következő fájlra. Amennyiben egy **REQUEST:fájlnév** típusú üzenet jött, akkor a szerver kéri a fájlt. Beolvassuk teljesen a fájlt a DataInputStream readFully műveletével a fájl méretének nagyságával egyenlő bytearray-ba, majd elküldjük a méretét és utána magát a fájlt (bytearray-t). A stopClient hívás esetén a running változó hamisra állításával az iteráció megszakítódik, illetve az iteráció végén is a kliens egy **Disconecting** üzenettel jelzi a szerver felé a lecsatlakozását, amit mindkét oldalon a felhasználói felületen is feltüntetünk.

4.7.4. A Server osztály



17. ábra A Server osztály

A Server osztály (lásd 17. ábra) konstruktorában megkapja a FileTransferUISetter példányát és az adott útvonalat. A **getTargetDirectory** és **setTargetDirectory** metódusokkal a fájlok másolásának helyét tudjuk megváltoztatni. Az **isServerRunning**-al a főmenüre való visszatérés elött ellenőrizzük, hogy a szál még fut-e, és ha igen megállítjuk. A **startServer** és **stopServer** metódus a változó nevekben tér el a

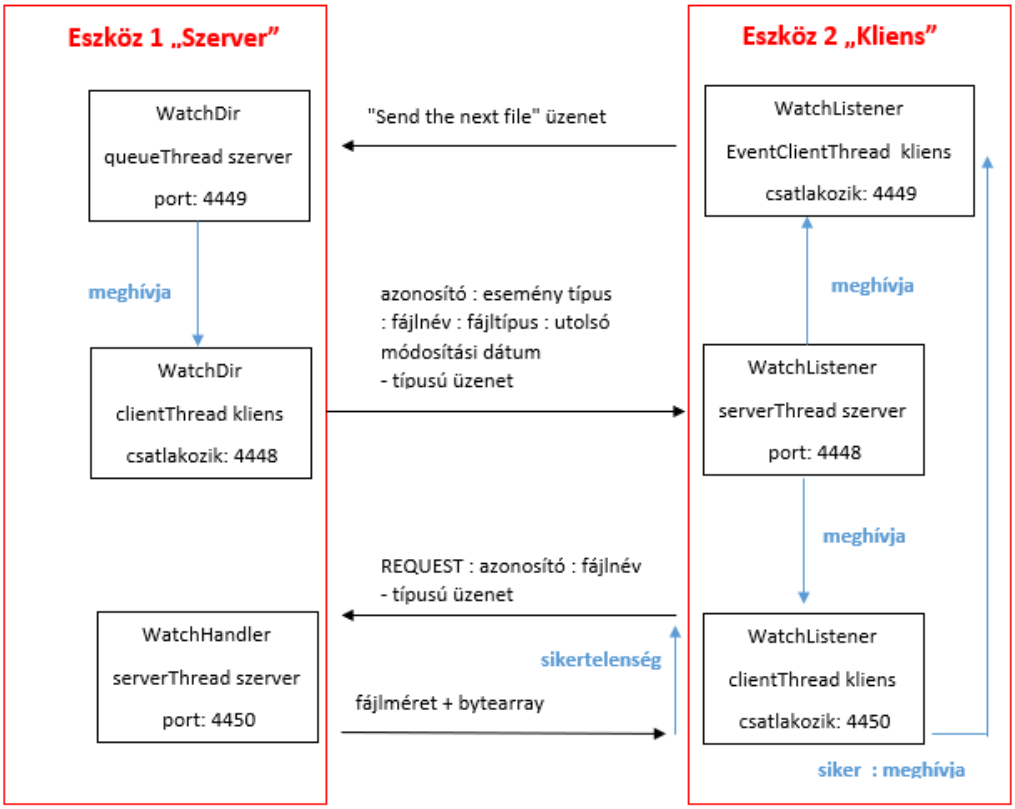
HomeCloudNetworking szerver részében megtalálható startServer és stopServer metódusoktól, azonkívül ugyanazt a célt és működést szolgálja. A **discoverDirectory** metódus itt is megtalálható annyi különbséggel, hogy az útvonalnál a paraméter helyett a targetDirectory-t használjuk, illetve itt már nem kell az elements-hez adnunk az elemeket.

A **createTCPServerThread** [3] metódus a szerver oldali kommunikációért felel. Az adott 4460-as porton létrehozzuk a TCP Socket szervert, majd, ha egy kliens csatlakozni szeretne, elfogadjuk, és a csatlakozást feltüntetjük a felhasználói felületben, illetve létrehozzuk a szükséges Data-Input/Output-Stream csatornákat. Amennyiben a csatlakozott klienstől a **Disconecting** üzenetet kaptuk, bezárjuk az adott klienssel a kapcsolatot, másképp értelmezzük és elemezzük az üzenetet. Mivel a kliens rákérdez minden egyes mappában található fájlra és ha szükség van rá, akkor elküldi a fájlt, így annak ellenőrzéseképp, hogy megvan, illetve a legfrissebb az adott fájl, szükség van az adott célmappa minden iterációban történő bejárására. Ezután ellenőrizzük az igényt a fájlra, amiben két esetet különböztetünk meg: már létezik, vagy még nem az adott fájl. Amennyiben létezik, akkor frissítésről beszélünk, mivel csak azt kell megnéznünk, hogy a kliens által küldött fájl frissebb-e. Amennyiben igen, elküldjük az igényt a fájlra, máskép kérjük a következő fájlt. A másik eset, ha a fájl nem létezik, akkor első sorban ellenőrizzük, hogy a könyvtár, amiben van, létezik-e. Ezt úgy tudjuk megnézni, hogy az adott fájlnév a split során több részre osztható mint 3. Például, ha a kapott fájlnév: **/a/a.txt**, akkor a String tömbünk 0. eleme üres, az 1. eleme az **a** és a 2. eleme pedig az **a.txt.** Amennyiben az elérési úton mappákat kell létrehozni fájl fogadás elött, a fájlnév nélkül összerakjuk az elérési útvonalat, majd létrehozzuk a mappákat az adott útvonalon. Végezetül elküldjük az igényt a fájlra. Abban az esetben, amikor igényt küldtünk, akkor várunk a kliens válaszára (a méretre és a bytearray-re). Ha megjött, akkor létrehozzuk az output fájlt, beolvassuk a fájl méretét, majd sorra egy buffer-be olvassuk a klienstől jövő üzenetet addig, amíg van mit olvasni, a bufferból pedig kiírjuk a fájlba. Ezután ellenőrizzük, hogy a fájl valóban megérkezett: ha igen, beállítjuk a kliens által küldött utolsó módosítási dátumot, ugyanis kiírás után a fájl az akkor aktuális dátumot állítja be utolsó módosítási dátumnak. Ezután a felhasználói felületben azonnal megjelenítjük, ha megérkezett a fájl, ellenkező esetben újra elküldjük az igényt a fájlra. Abban az esetben, ha a kliens "Disconecting" üzenetet

küldött vagy pedig megállítottuk már a szervert, lezárjuk a klienssel való kapcsolatot. Ha lezártuk a szervert, akkor a felhasználói felületben is feltüntetjük.

4.8. Directory listener

A Directory listener funkció valósítja meg az adott mappa figyelését és egy benne lévő fájl/mappa változásra történő átküldését. Hasonlóan a File Transfer-hez már tudják az eszközök a másik IP címét, ezért a fájl átvitelhez ugyancsak a TCP protokollt használjuk. Annak érdekében, hogy könnyebben megértsük a két eszköz között hogyan valósul meg a kommunikáció, a 18. ábra reprezentálja a két fél közötti üzenetváltásokat.



18. ábra A Directory listener funkció kommunikációs diagramja

A Directory listener funkció 8 részből áll: MessageQueue, WatchDir, WatchDirectory.fxml, WatchDirectoryController, WatchDirectoryUI, WatchHandler, WatchDirectoryUIsetter, WatchListener. Ahogy az eddigi fxml fájlok, úgy a WatchDirectory.fxml fájl is a Scene Builder segítségével került összeépítésre, kinézete pedig majdnem azonos a FileTransfer.fxml-el. A WatchDirectoryUI (lásd 19. ábra) az fxml fájl betöltéséért felel, de tartalma lényegében ugyan az mind a FileTransferUI-é. WatchDirectoryUIsetter interfész (lásd 19. ábra) a felhasználói felület WatchDir,

WatchHandler és WatchListener osztályokból való módosításáért felel futás közben.

4.8.1. A WatchDirectoryController osztály

19. ábra A WatchDirectoryUI és WatchDirectoryController osztályok, valamint a WatchDirectoryUIsetter interfész

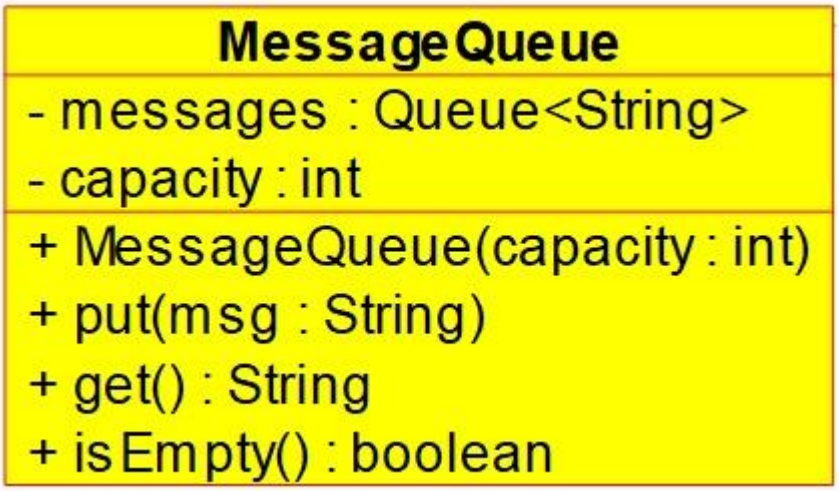
A WatchDirectoryController osztály (lásd 19. ábra) szolgál a felhasználói felület funkcionalitásáért. A metódusok nagyjából megegyeznek a FileTransferController-ben levőkkel. A **changeAppMode**-ban a különbség, hogy más az ablakra való kiírt szöveg, illetve a **Check failed files** gombot a **Start server** gombbal egyszerre aktiváljuk és deaktiváljuk. A **handleBackToMenu** is azonos azzal a különbséggel, hogy Servernek a WatchHandler-t tekintjük, Client-nek pedig a WatchListener-t, így a főmenüre való visszatérés elött ezek futását ellenőrizzük. Az **UILogger** és **chooseDirectory** metódusok teljesen azonosak a FileTransferController-ben levőkkel. A **handleStartStopServer** és **handleStartStopClient** szinte teljesen azonos azzal az eltéréssel, hogy első a server nevet kapja a WatchHandler példányosításakor és ennek az elindításáért felel, a második a client nevet kapja a WatchListener példányosításakor és ennek az elindításáért felel. Mindkét metódus azon kívül, hogy egymással azonosak, a FileTransferController-ben levő handleStartStopServer-rel is megegyeznek azzal az eltéréssel, hogy a felhasználói felületben feltüntetjük hogyha már elindítottuk

mindkettőt. A **handleCheckFailedFiles** metódus, amennyiben a WatchHandler már létrehozásra került, meghívja annak a **WatchHandlerCheckFailedFiles** metódusát, amivel majd a sikertelen fájlokat akarjuk ellenőrizni.

4.8.2. A WatchDirectoryUIsetter interfész a WatchDirectoryController osztályban

A WatchDirectoryUIsetter példányosítása során a **FailedFilesChecked**, **serverStarted** és **clientStarted** metódusokat a felhasználói felület státusz üzenetének frissítésére használjuk rendre akkor, ha ellenőriztük a sikertelen fájlokat, elindult a WatchHandler és ha elindult a WatchListener. A **setListViewToWatchHandler** és **setListViewToWatchListener** metódusok azonosak azzal a különbséggel, hogy egyik a Server Mode-ban levő eszköznek, a másik a Client Mode-ban levő eszköznek állítja a ListView-ját. A metódusok a **FileTransferController**-ben levő **setListViewToServer**-re hasonlítanak, de ezen kívül még a **stopWatchListenerUI** is lényegében megegyezik a **clientStopped** metódussal. A **stopWatchHandlerUI** során a WatchHandler-t akarjuk megállítani. Ahhoz, hogy ez bekövetkezzen a hívást 3-szor kell megtenni: egyszer a WatchHandler **createTCPServerThread** metódusának végeztével, továbbá a WatchDir **processEvents** és **watchQueueServerThread** metódusainak végeztével. Amikor mindhárom helyről megtörtént a stopWatchHandlerUI meghívása, csak akkor lesz ismét elindítható a WatchHandler. Annak érdekében hogy a hívások szinkronizált módon történjenek, a **synchronized(WatchHandlerStatusLock)** hívással oldjuk meg, ahol a WatchHandlerStatusLock egy Object.

4.8.3. A MessageQueue osztály



20. ábra A MessageQueue osztály

A Directory listener során, amikor figyelünk egy adott mappát, egy elem behúzásakor a WatchDir-ben levő WatchService eseményeket generál. Ezeket az eseményeket kell majd küldjük a másik eszközön futó szervernek a WatchListener-ben. Ez a példa lényegében a termelő-fogyasztó probléma esete. A termelő a maga ütemében termel, és ha végzett kiteszi a polcra, majd a fogyasztó ugyancsak a maga ütemében a terméket meg veszi le a polcról. Fennállhat az az eset, amikor a termelő túl gyorsan termel, és telerakja a polcot, ekkor várnia kell addig amíg a fogyasztó helyet szabadít a polcon. Ez az eset fordítva is fennállhat: a fogyasztó túl gyorsan veszi le a terméket, ezért a polc megürül, így várnia kell addig amíg új termék kerül a polcra. Hogyha a mi esetünkre vesszük a fentebb említett példát, akkor észrevehetjük a hasonlatot, hogy a termelő esetünkben az esemény generáló WatchDir, a termékek az adott események, amik egy fájl/mappa behúzása során generálódnak, a fogyasztó pedig a WatchListener, ami feldolgozza az eseményeket. A probléma megoldásának kivitelezéséhez mindenek elött szükségünk van egy olyan adatszerkezetre, ami a fenti példában a polcot valósítja meg. Ennek kivitelezéseként funkcionál a **MessageQueue** [6] osztály (lásd 20. ábra). Konstruktorában egy számot kap, ami az adott polc vagy Queue méretét határozza meg. A **put** metódus megvalósítja a példánkban azt, amikor a termelő addig tesz bele a sorba adatot, ameddig el nem éri a maximum kapacitást. Ekkor várakozásba kezd, hogy a fogyasztó szabadítson fel helyet. Minden alkalommal amikor beleteszünk egy adatot a sorba, értesítsük a fogyasztót, mert lehet, hogy nincs

adat a sorban, és a fogyasztó pont az adatra vár. A **get** metódus a másik esetet valósítja meg, amikor a fogyasztó adatot vesz ki a sorból. Ha a sor üres, akkor addig vár ameddig a termelő adatot nem tesz bele. Ha kivettük az adatot, értesítsük a termelőt, mert lehet, hogy tele van a sor és arra vár, hogy hely szabaduljon fel. Végezetül a sorból kivett értékkel térünk vissza. Mivel a get metódus aktívan blokkol addig, amíg adat nem kerül a sorba, meghívása elött majd az **isEmpty** metódust hívjuk meg. Ez megmutatja, hogy a sor üres vagy nem, így ha üres, akkor nem hívjuk meg fölöslegesen a get metódust, ami által megszüntettük a blokkolását.

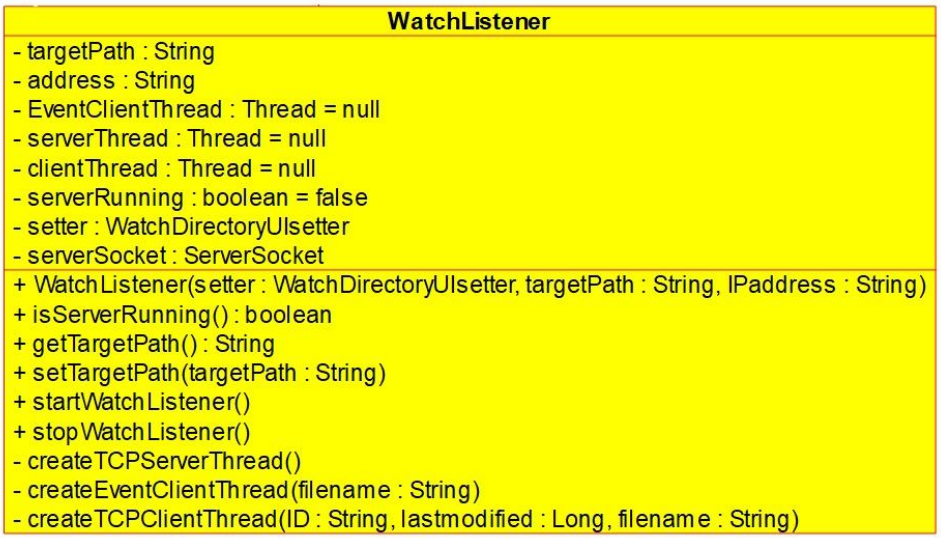
4.8.4. A WatchHandler osztály

21. ábra A WatchHandler osztály

A WatchHandler osztály (lásd 21. ábra) konstruktorában paraméterként megkap egy WatchDirectoryUIsetter példányt, az adott útvonalat és a másik eszköz IP címét. Az **isRunning** metódusával tudjuk meg, hogy fut-e, meghívására a főmenüre való visszatérés elött kerül sor. Mivel két részből tevődik össze a futása, (önmaga és a WatchDir serverRunning-ja) emiatt két érték együttesével térünk vissza. A **getFilePath** és **setFilePath** metódusokkal tudjuk változtatni az elérési útvonalat. A **WatchHandlerCheckFailedFiles** metódus a WatchDir-ben levő **FailedFiles** metódus hívásával ellenőrzi a sikertelen fájlokat. A **startWatchService** metódus a WatchHandler elindításáért felel: ha mar létezik a működésért felelős szál, akkor return-olunk. Minden indításnál új WatchDir-t hozunk létre, aminek átadjuk Path-ként a megadott elérési útvonalat és az IP címet, majd ezt követően el is indítjuk, végül létrehozzuk az

itteni szerverért felelős szálat. A **stopWatchService** metódus a WatchHandler megállításáért felel: ha nem létezik még a WatchDir példány, akkor return-olunk, másképp a WatchDir-t állítjuk meg először. Mivel a ServerSocket accept metódusa blokkoló jellegű, emiatt ennek feloldásához szükség van a ServerSocket.close hívásra a szerver lezárásához. A **createTCPServerThread** metódusban levő serverThread szál felel a Directory listener során a fájlok beolvasásáért és átküldéséért. Elve a File transfer Client osztályához megegyező. Megnyitunk egy ServerSocket-et a 4450-ös porton, és várjuk, hogy a másik eszköz kliense csatlakozzon, majd ezzel létrehozzuk a Data-Input/Output-Stream csatornákat. A kapott üzenet három részből kell összetevődjön: **REQUEST:ID:fájlnév**. Ez jelzi, hogy a másik eszköz kéri az adott eseményben létrehozott/módosított fájlt. A továbbiakban a File transfer Client-ben levő módon már csak szimplán létrehozzuk a fájl méretével megegyező méretű bytearray-t, és az üzenetben kapott nevű fájlt a DataInputStream readFully metódusával teljesen beolvassuk, majd válaszként elküldjük a fájl méretét és magát a bytearray-t. Amennyiben az beolvasás során a fájl nem létezik, a fájl mérete helyett -1 -et küldünk. Végezetül lezárjuk a kapcsolatot a klienssel. Megszakítás során lezáródik a szerver, amit a felhasználói felületben feltüntetjük azáltal, hogy meghívjuk elsőként a háromból a stopWatchHandlerUI metódusát a WatchDirectoryUIsetter interfésznek.

4.8.5. A WatchListener osztály



22. ábra A WatchListener osztály

A WatchListener osztály (lásd 22. ábra) valósítja meg egyik eszköz fogadó felét,

vagyis a Client Mode-ban való használatát. Hasonlóan, konstruktorában paraméterként megkap egy WatchDirectoryUIsetter példányt, az adott útvonalat és a másik eszköz IP címét. Az **isServerRunning** metódusával tudjuk meg, hogy fut-e, meghívására a főmenüre való visszatérés elött kerül sor. A **getTargetPath** és **setTargetPath** metódusokkal tudjuk változtatni az elérési útvonalat. Elindításáért a **startWatchListener** felel, melyben, ha már létezik a felelős serverThread szál, akkor return-olunk, ha még nem, akkor létrehozzuk. A **stopWatchListener** a megállításért felel, hasonlóan, ha nem létezik még a felelős szál, akkor return-olunk. A ServerSocket accept metódusa itt is blokkol, így a ServerSocket.close hívással zárjuk le.

A **createTCPServerThread** metódusban levő serverThread szál felelős a szerverként való működésért. A ServerSocket szervert a 4448-as porton üzemeltetjük, ha egy kliens csatlakozott fogadjuk. Üzenetben a másik eszköztől a már ott feldolgozott eseményeket kapjuk meg. Az üzenet tartalmazza az esemény azonosítóját, fajtáját, fájlnevét, fájltípusát és az utolsó módosítási dátumot. Első sorban megnézzük, hogy az üzenet teljes-e, ENTRY_CREATE vagy ENTRY_MODIFY típusú esemény-e, amit kaptunk. A továbbiak működésében a File transfer Server-jéhez hasonlóan két esetet különböztetünk meg: megvan a fájl vagy nincs. Ha megvan, akkor a fájl frissítése történik, másképp kérjük a következő eseményt. Ha nincs meg a fájl, akkor először megnézzük, hogy mappa-e. Ha igen, akkor létrehozzuk, és kérjük a következő eseményt. Azt is ellenőriznünk kell, hogy a fájl esetleg nemlétező almappákban van-e, mert ha igen, akkor küldés elött azokat kell létrehozni, ha még nem léteznek, csak ezután történhet meg a fájl küldése. Végezetül bezárjuk a klienst és feltüntetjük a szerver zárását a felhasználói felületen.

Az osztályban továbbiakban még két kliens szerepel. A **createEventClientThread** feladata, hogy megvalósítsa a EventClientThread szálat, ami létrehoz egy Socket klienset és csatlakozik a másik eszközön levő 4449-es porton futó szerverhez. Csupán egy üzenetet küld el, amivel jelzi, hogy végzett a megadott fájl átküldésével vagy feldolgozásával és ezzel kéri a szervertől a következő eseményt. Amikor mappa létrehozás történt, vagy ha nincs mit kezdjünk az adott eseménnyel, akkor az üzenet első része üres lesz, másképp a fájl neve fog szerepelni.

A **createTCPClientThread** metódusban létrehozzuk a clientThread nevezetű szálat, melynek működése hasonló a File transfer Server-jéhez. A paraméterből kapott

adatokkal elküld egy **REQUEST:ID:fájlnév** típusú üzenetet, amivel kéri az adott fájlt a másik eszköz szerverétől. Miután elküldtük a kérést, várunk a szerver válaszára, vagyis az adott fájlra. Beolvassuk a fájl méretét, majd a fájlt reprezentáló bytearray-t a DataInputStream csatornából, végezetül pedig kiírjuk a fájlt. Amennyiben -1 jött a fájl méretének, kilépünk. Végezetül leellenőrizzük, hogy tényleg megérkezett-e a fájl, és ha igen, akkor beállítjuk a módosítási dátumot. Amennyiben nem jött át a fájl, az egész műveletet addig ismételjük, amíg meg nem érkezik. Ezt követően bezárjuk a klienset és a csatornákat, majd kérjük a következő eseményt.

4.8.6. A WatchDir osztály

23. ábra A WatchDir osztály

A WatchDir osztály (lásd 23. ábra) felel az adott mappa figyeléséért. Konstruktorában paraméterként megkap egy WatchDirectoryUIsetter példányt, az adott útvonalat Path típusként és a másik eszköz IP címét. Inicializáljuk a WatchService-t, illetve két MessageQueue-t. Annak érdekében, hogy regisztráljuk figyelésre a mappákat, meghívjuk a **registerAll** metódust. A metódusban a paraméterként

megadott elérési utat (az adott könyvtárat, amit figyelni fogunk) és a benne levő alkönyvtárakat járjuk be, és regisztráljuk őket a WatchService-el, azaz eltároljuk őket a keys nevű Hashmap-ben. Négy eseményt figyelünk: fájl/mappa törlés, létrehozás, módosítás és esemény túlcsordulás. Ezáltal tudni fogjuk, hogy ha a felsorolt események közül valamelyik bekövetkezett a megadott elérési úton. A metódus rekurzívan hívja önmagát, így a kijelölt elérési utat és a benne levő almappákat teljesen bejárja. Az **isRunning** metódussal kérdezhető le, hogy a mappa figyelés és az esemény feldolgozás fut-e. A megállításáért a **stopWatchDir** felel, melyben a queueThread szálat figyeljük, hogy létezik-e már, mert ha nem, akkor return-olunk. A ServerSocket accept hívása a **watchQueueServerThread** metódusában blokkoló jellegű, emiatt ennek feloldásához szükség van a szerver lezárásához a ServerSocket.close hívással.

A **processEvents** [5] metódus a Directory listener funkció legfontosabb metódusa, ugyan is ez felel az események figyeléséért. Ha már létezik a szál, akkor return-olunk. A futást igazoló running változót igazra állítjuk, továbbá meghívjuk a **watchQueueServerThread** és **handleEventsStorageThread** metódusokat (amik működését a későbbiekben fejtek ki részletesen), melyek csak röviden szólva létrehoznak 1-1 szálat: az első a megtörtént eseményeket tárolóként fogadja, majd feldolgozza őket, azután a kész küldhető eseményeket a másik szál tárolójába helyezi, ami aztán az események másik eszközhöz való eljuttatásáért felel. Visszatérve a processEvents-re, az események feldolgozása során először lekérünk egy watchkey-t a poll hívással, mivel ez azonnal null-al tér vissza, ha nem érhető el a kulcs, és így nem blokkol. Ha a kulcs, amit kaptunk nem volt regisztrálva inicializálás során, akkor eldobjuk és folytatjuk a ciklust, másképp pedig feldolgozzuk a függő eseményeit. Nem foglalkozunk a következő esetekkel: amennyiben OVERFLOW vagy ENTRY_DELETE történt (túlcsordulás vagy törlés), vagy pedig egy esemény már nem először fordult elő. Az esemény során érintett fájl nevét/útvonalát az event context-ből tudjuk kinyerni, de ehhez annak WatchEvent<Path> típusúnak kell lennie, így emiatt, hogy ne kapjunk szimpla cast-olással *unchecked or unsafe operations* hibaüzenetet, szükség van a statikus cast metódusra, ami elfolytja nekünk az *unchecked* hibát. Így, hogy megkaptuk az aktuális esemény útvonalát meg kell néznünk, hogy ENTRY_CREATE (új fájl/mappa megjelenése) esetén mappa volt-e, ami létrejött, mert ha igen, akkor azt is be kell regisztrálnunk. Az esemény feldolgozás végén pedig egy **ID~esemény típus~fájl**

útvonal típusú üzenetet rakunk be az eventsStorage tárolóba, ahol majd a eventsStorageThread szálban feldolgozásra kerülnek. Végezetül újraindítjuk a watchkey kulcsot, mivel már feldolgoztuk a megtörtént eseményeit, majd kitöröljük a keys Hashmap-ből, ha a mappa már nem elérhető többé. A processEvents végeztével a WatchService-t és a keys Hashmap-et null-ra állítjuk és meghívjuk második alkalommal a stopWatchHandlerUI-t, a WatchHandler megállításához.

A **handleEventsStorageThread** metódus az események tárolásáért és feldolgozásáért felel. A megvalósítás során létrehozunk egy eventsStorageThread nevezetű szálat, amiben egy while ciklust futtatunk. Ha az eventsStorage tároló üres, akkor várakozunk azáltal, hogy 1 milliszekundumot altatunk, másképp, ha egy esemény történt, akkor kivesszük az eventsStorage tárolóból. A Directory listener során megesett, hogy nem tudtuk beállítani sikeresen az utolsó modifikálási dátumot mivel az eredeti dátum helyett a fájl átvitel aznapi és akkori dátuma szerepelt. Ennek oka, hogy a Windows a fájlokat részcsomagokban küldi át, amely során a fájl tulajdonságait alapértelmezettre állítja. Miután a fájl átküldésre került, frissíti a tulajdonságait. Az utolsó módosítási dátum ismerete nélkül, az alapértelmezett érték az akkori idő lesz. A probléma ebből adódott, hogy mikor behúztuk a fájlokat a figyelt mappába, a Windows elkezdi másolni a fájlokat, amit a WatchService észrevesz és elküldi az eseményt. Amikor létrehozzuk a new File() példányt megeshet, hogy a fájl már megérkezett, viszont a tulajdonságai még nem lettek frissítve, mivel még ezek beállítása előtt értük el. Emiatt az f.lastModified() hívás az akkori dátumot fogja visszaadni, és ezzel fogunk dolgozni. Hogy a fenti esetet elkerüljük, a new File() példányosítása előtt 200 milliszekundumot altatunk, hogy legyen idő a fájl tulajdonságainak beállítására. Ezt követően létrehozzuk a File példányt és először megnézzük, hogy az utolsó modifikálás dátuma 0-e, mert ha igen, az azt jelenti, hogy a fájlt vagy mappát törölték, így nem kell foglalkoznunk vele. Az elérési útból kinyerjük a fájl nevét, majd ellenőrizzük, hogy fájl vagy mappa-e. Ha mappa, akkor hozzáadjuk a **FailedFiles** metódusban használatos mappa elérési útvonalakat tartalmazó listához. Elkészítjük az esemény üzenetet, ami egy ID-ból, esemény típusból, fájlnévből, fájltípusból és utolsó modifikálási dátumából áll. Ha ez az első esemény, akkor már itt elküldjük a másik eszközhöz, másképp betesszük a már feldolgozott és küldhető eseményeket az mq nevezetű tárolóba. Végezetül megjelenítjük a felhasználói felületen az eseményt.

A **watchQueueServerThread** metódus az események kiadásáért felel. Egy szervert futtatunk a 4449-es porton, amivel várjuk a másik eszköztől jövő üzenetet, hogy küldhetjük a következő eseményt, mivel csak akkor küldhetünk egy eseményt, ha a másik fél már feldolgozott egyet, és kéri a következőt (kivételt képez ez alól a legelső esemény). A klienstől kapott üzenet **fájlnév:Send the next file** jellegű kell legyen. Ha az első rész üres, akkor mappa létrehozás történhetett, vagy csak egyszerűen nincs szükség az adott eseményre (például a fájl már létezik). Ha nem üres az első rész, akkor eltároljuk mint átküldött fájlt, és ha a második rész a **Send the next** file üzenetet tartalmazza, akkor kivesszük a már küldhető eseményeket tartalmazó mq tárolóból és elküldjük a másik eszköznek a **createTCPClientThread** metódus hívásával. Amennyiben a tároló üres, egy while ciklusban a Thread.sleep metódussal altatunk 1 milliszekundumot mindaddig amig a tárolóba lesz érték. Végezetül itt kerül meghívásra a stopWatchHandlerUI harmadjára, ezáltal a WatchHandler véglegesen leállt, így a Start Server gomb elérhetővé válik.

A **createTCPClientThread** metódus az esemény másik eszközhöz való eljuttatásáért felel. Ha már létezik a szál, akkor return-olunk, egyébként létrehozunk egy Socket-et és felcsatlakozunk a másik eszköz 4448-as porton levő szerverére, es elküldjük neki a történt eseményt, amit a msg paraméter jelöl.

A **FailedFiles** metódus a sikertelen fájlok elküldésének ellenőrzéséhez szolgál, amikor az ablakban megnyomjuk a **Check failed files** gombot kilépés elött. Azért van erre szükség, mert néhány esetben a WatchService-el megtörténhet, hogy nem veszi észre az új mappa beillesztésekor a benne lévő összes fájlt. Ezért lényegében hogyha valamilyen esemény történik, amely egy mappát érint, eltároljuk az adott mappa elérési útvonalát. Amikor a felhasználó végzett a fájl átvitelével, a gombnyomásra az összes érintett mappát bejárjuk, és kigyűjtjük a bennük lévő fájlokat egy teljes listába. A WatchDir-ben mindig eltároltuk egy listában hogyha egy fájl átvitelre került, így a következő lépésben bejárjuk az átküldött fájlok listáját, és ha az adott fájl már átküldésre került, akkor kivesszük a mappák bejárásából létrehozott teljes listából. Végezetül pedig ami a teljes listában maradt, azok az át nem küldött fájlok. Bejárjuk a listát, és a fájlokat betesszük friss létrejött eseményekként az eseményeket tároló eventsStorage listába, majd megjelenítjük a felhasználói felületen, hogy leellenőriztük a sikertelen fájlokat.

5. Fejlesztői dokumentáció – HomeCloudMobile

Az alkalmazás Androidra való portolása során az asztali alkalmazás forráskódja újbóli felhasználásra került, azonban az Android másfajta preferenciái és jellege miatt szükség volt pár alkotó elem megváltoztatására. Ebben a részben az Android verzió, asztali alkalmazástól eltérő részeit mutatnám be.

5.1. Fejlesztőeszközök

Az asztali verzióhoz hasonlóan, az Android alkalmazás ugyan úgy a Java nyelvet használja. Fejlesztői környezetnek az IntelliJ szoftverjére épülő Android Studio-t, projektépítő eszköznek pedig a vele használatos Gradle-t választottam. A JDK helyett, itt az Android **Software Development Kit**-et (SDK) használjuk, de az Android Studio már tartalmazza ezt, így csak az Android Studio-t kell letölteni, ezen kívül nincs más teendőnk a fejlesztés kezdete előtt.

Az Androidra való portolás előtt első lépésként szükség volt a JavaFx elemek eltávolítására. Mivel az Android lényeges elemei az úgynevezett Activity-k és Intent-ek, az osztályokat és metódusokat ennek megfelelően kellett módosítani. Az Activity-ket legkönnyebben úgy lehet elképzelni, mint egy ablakot, ugyanis az Androidon levő oldal/lapokat reprezentálja, az Intent pedig a Activity-k létrehozásáért felel. Az asztali alkalmazás .fxml fájljainak szerepét a .xml fájlok veszik át, ezek fogják tartalmazni a felhasználói felület elemeit és kinézetét. Az UI és Controller osztályok összeolvadtak, és ezek lettek az Activity osztályok, melyek az adott xml fájlt betöltik, és itt fejtjük ki az ablakok funkcionalitásait. A Scene Builder-hez hasonlóan, az Android Studio lehetőséget ad az .xml fájlok drag and drop módszerrel való elkészítéséhez.

5.2. A főmenü

Az alkalmazás megnyitásakor egy üdvözlő lap fogad minket. A megvalósításért a MainActivity [7] osztály felel: a felülírt **onCreate** metódusban betöltjük az activity_main.xml fájlt. A fájl lényegében egy szín grádiens [8 , 9] háttérrel ellátott, két sor szöveget tartalmazó ablak. A Handler.postDelayed hívással 4000 milliszekundum múlva létrehozzuk egy Intent-el a HomeActivity nevű Activity-t és el is indítjuk, majd befejezzük az aktuálisat.

A főmenüért a fentebb említett HomeActivity osztály felel. Az **onCreate**

metódusában betöltjük az activity_home.xml fájlt, majd letiltjuk a **START DISCOVERY**, **WATCH A DIRECTORY** és **FILE TRANSFER** gombokat. Utána az Appmode-ért felelős Switch-nek beállítjuk és felül írjuk az **onCheckedChanged** metódusát, ami majd a Server és Client mód között való csúsztatásért felel, változtatva az Appmode feliratát, aktiválva és deaktiválva a SERVER és DISCOVERY gombokat. Végezetül meghívjuk a **checkPermission** [16] metódust, ami ellenőrzi nekünk, hogy az alkalmazás tárhelyhez való engedélye meglett-e adva, és ha nem, akkor kérelmezi. A felülírott **onRequestPermissionsResult** callback metódusként hívódik meg az engedélykérés eredményének ellenőrzés során. Ha az engedélyt megkaptuk, akkor **Storage Permission Granted** Toast üzenetet látjuk, ha pedig nem, akkor kilépünk.

Az Overflow menüért [10] a felülírott **onCreateOptionsMenu** metódus felel, betölti a menu_items.xml fájlt, amiben az Overflow menü Help és Exit eleme található. Ezek kiválasztásakor történt eseményekért a felülírott **onOptionsItemSelected** metódus felel. Ha a Help-et választottuk, akkor egy Intent-el létrehozzuk a Help nevű Activity-t és el is indítjuk. A Help Activity a Help ablakért felel, csak a felülírott **onCreate** metódust tartalmazza, ami betölti az activity_help.xml fájlt. Ha az Exit-et választottuk, akkor először ellenőrizzük, hogy példányosítottuk-e már a HomeCloudNetworking osztályt, és ha igen, akkor fut-e a szervere és kliense. Ha igen, akkor bezárjuk őket, utána befejezzük az aktuális Activity-t, majd egy System.exit hívással kilépünk.

A továbbiakban egy nagyon fontos eltérés az, hogy az Android a felhasználói felületen levő elemeket View-ekként, azaz nézetekként kezeli. Minden nézetnek megvan a saját maga azonosítója az xml fájlban, amire hivatkozni a **findViewById(R.id.AZONOSÍTÓ_NEVE)** hívással tudunk, amit majd cast-olhatunk arra ami nekünk szükséges. Ennél az okból adódóan, egy gomb funkcionalitásáért felelős metódus publikus kell hogy legyen, és paraméterként mindig egy View-et kap, ami az adott gombot/elemet reprezentálja. Így a továbbiakban mindig gombbá cast-oljuk a View-et, majd például szövegét lekérjük, és azzal dolgozunk tovább, ahogy ezt a **handleStartStopServer** és **handleStartStopDiscovery** metódusokban is tettük, ezáltal a gomb elérésén kívül az eddigi metódusok változatlanul maradtak. A **handleWatchDirectory** és **handleFileTransfer** metódusok Intent-el hozzák létre a WatchDirectoryUI és FileTransferUI Activity osztályokat, és az eddig paraméterül kapott másik eszköz címét, az osztály konstruktorának paraméterül adása helyett pedig

a intent.putExtra hívással adjuk át. A felhasználói felületen futás közben való változtatásért, a Platform.runLater hívás helyett egy másfajta módszert [12] kellett alkalmaznunk. Létrehozunk egy Handler-t, majd annak a postDelayed hívására az első paramétereként kap egy Runnable objektumot, aminek felülírjuk a run metódusát, és abban fejtjük ki a változtatásokat, majd a második paraméterként megadjuk, hogy 100 milliszekundum késleltetés múlva hajtódjon végre. Ezt a megoldást használjuk a **UILogger** metódusban is.

5.3. Network Discovery

A Network Discovery során maga az Android koncepciójának jellege miatt volt szükség egy kis kód hozzáadására. A probléma [13] onnan adódik, hogy a WifiManager.MulticastLock lehetővé teszi egy alkalmazás számára, hogy Wifi Multicast csomagokat küldjön és fogadjon. Normális esetben az úgynevezett Wifi verem kiszűri azokat a csomagokat, amelyek nem kifejezetten annak az eszköznek vannak címezve. A MulticastLock megszerzése teszi lehetővé, hogy a verem Multicast címekre címzett csomagokat tudjon fogadni. Emiatt annak érdekében, hogy Multicast csomagokkal tudjunk dolgozni, meg kell szerezni ezt a MulticastLock-ot. Ehhez az AndroidManifest.xml fájlban deklarálnunk kell a következő engedélyeket: CHANGE_WIFI_MULTICAST_STATE, ACCESS_WIFI_STATE, ACCESS_NETWORK_STATE, INTERNET. Annak érdekében, hogy példányosítani tudjunk egy WifiManager-t, szükségünk van a Context.WIFI_SERVICE értékre. Emiatt, hogy ezt a Context értéket le tudjuk kérni, szükség volt arra, hogy a HomeCloudNetworking osztály is egy Activity legyen. Ezáltal felülírjuk az Activity-k **onCreate** metódusát, amiben hogyha példányosítani tudtunk egy WifiManager-t, akkor létrehozunk egy MulticastLock példányt, majd az acquire hívással megszerezzük a MulticastLock-ot.

5.4. Watch a directory és File transfer

A **Watch a directory** és **File transfer** funkciók felhasználói felületének ablaka, és az ablak funkcionalitását megvalósító Activity osztályok majdnem azonosak, és mivel a **Watch a directory** funkciót megvalósító osztályok nem nagyon változtak, emiatt is egy fejezet alá vesszük a két funkciót.

Első sorban, mindkét ablak meghívásakor az Activity-k **onCreate** metódusát írjuk felül, amiben betöltjük az activity_filetransfer és activity_watchdirectory .xml fájlokat,

majd lekérjük a létrehozott Intent-et, és annak a getStringExtra metódusával megkapjuk a másik eszköz IP címét. A továbbiakban a főmenühöz hasonló inicializálás történik. Kezdésből deaktiváljuk a **START CLIENT** gombokat, majd az alkalmazás Server és Client Mode-ban való használatáért felelős Switch-nek beállítjuk és felülírjuk az **onCheckedChanged** metódusát. Ez a Server és Client Mode között való csúsztatásért felel, változtatva az Appmode feliratát, az ablakban található szövegeket, aktiválva és deaktiválva a gombokat. Továbbá az asztali alkalmazásban szereplő ListView-t egy ScrollView-be helyezett LinearLayout helyettesíti, így majd ebben fognak megjelenni az átküldött vagy talált fájlok, létrejött események.

A mappa választás [11] másképp történik, ugyanis egy Intent-et hozunk létre, és paraméterben megadjuk az Intent.ACTION_OPEN_DOCUMENT_TREE értéket. Ez fogja lehetővé tenni az adott típusú mappakiválasztó felületet, amit majd létre is hozunk. Annak érdekében, hogy megkapjuk a kiválasztott értéket/mappát, felül kell definiálnunk a **onActivityResult** metódust. Az Intent.getData hívással megkapjuk a kiválasztott mappa útvonalát URI formátumban. Ennek valós elérési útvonallá való alakításához a **findFullPath** [14] metódusra van szükségünk. Az URI elérési utat Path változóvá alakítjuk át, így ha a telefon tárhelyén kiválasztjuk mondjuk a Download mappát, akkor egy **/tree/primary:Download** típusú elérési utat kapunk. Azonban, ha az SD kártyán választjuk ki a Music mappát, akkor az útvonal, a primary kulcsszó helyett egy SD kártyát jelölő szám fog szerepelni, így **/tree/2D56-A326:Music** típusú lesz. Ahhoz, hogy dolgozni tudjunk vele, az elsőt **/storage/emulated/0/Download** típusúvá, a másodikat **/storage/2D56-A326/Music** típusúvá kell alakítani. Ezt a metódusban nem a leg optimálisabban oldjuk, ugyanis lényegében eldobjuk az első 5 betűt, helyette a /storage előtaggal elkezdjük építeni a String-et, és egy for ciklussal karakterenként bejárjuk, és az útvonalhoz hozzáadjuk a karaktereket addig, ameddig a : karakterrel nem találkozunk, mert ekkor egy / karaktert teszünk a helyére. Egy következő for ciklussal a : karakter utántól folytatjuk a bejárást a végéig. Ha primary szót tartalmazta az útvonal, akkor helyettesítjük egy /emulated/0/ -al, másképp csak szimplán visszatérünk a felépített String-el. Végezetül megjelenítjük a felhasználói felületen a megadott útvonalat, és a **START SERVER** és **START CLIENT** gombok csak innentől számítva lesznek elindíthatóak. A **Watch a directory** funkciót megvalósító osztályok metódusaiban a fentebb említett változások történtek. Annak érdekében,

hogy a metódusok ugyan úgy, Android 8.0-ás verzió feletti készüléken is futni tudjanak, a következő metódusok elé szükségszerű volt a **@RequiresApi(api = Build.VERSION_CODES.O)** annotáció beszúrása:

- WatchDirectoryUI: handleStartStopClient, handleStartStopServer, handleCheckFailedFiles
- WatchListener: startWatchListener, createTCPServerThread
- WatchHandler: WatchHandlerCheckFailedFiles, startWatchService
- WatchDir: cast, WatchDir, registerAll, processEvents, FailedFiles

Annak érdekében, hogy a **File transfer** funkciót 5.0-ás Android verziótól lehessen használni, minimális változtatásokat kellett végezni, például a Server osztályban, amikor azt ellenőrizzük, hogy a fájl almappákban van-e, és azokat létre kell-e hozni, az almappák létrehozását a File.mkdirs hívással oldjuk meg. A másik változtatás a mappák bejárását érintette a Server és Client osztályokban. A discoverDirectory annyiban változott, hogy a Files.walk metódus helyett a getListFiles [15] metódust használjuk. Ez lényegében a File.listFiles metódussal kilistázza, mi van az adott mappában, az értékeket eltároljuk egy File tömbben, amit bejárunk, és ha az adott elem egy mappa, akkor rekurzívan újra hívjuk a metódust. Ezen aprócska módosításokkal a funkció 5.0-ás Android verziótól is használhatóvá vált.

6. Tesztelés

A tesztelés során a meglévő két funkciót fogjuk ellenőrizni. A File transfer során azt vizsgáljuk, hogy a kijelölt mappában levő valós fájlokat (üres mappák nélkül) át tudjuk-e küldeni a kijelölt tároló helyre. A Directory listener során pedig azt vizsgáljuk, hogy a kijelölt mappa észreveszi-e a behúzott fájlokat, majd leszinkronizálja-e őket a kijelölt helyre. Ugyanakkor azt is vizsgáljuk, hogy amennyiben vannak olyan fájlok, amiket nem sikerült átküldenie, a Check Failed Files gombnyomás után át lesznek-e küldve. A teszt során ez úgy fog megnyilvánulni, hogy teszt könyvtárakat veszünk, és a bennük lévő fájlokat fogjuk másolni és figyelni. A teszteket több eszköz között is ellenőrizzük. A teszteléshez 2 mappát fogunk használni: to és from. A to mappát a fájlok eltárolásához használjuk, így majd ebben ellenőrizzük, hogy a fájlok megjöttek-e. A from mappát használjuk a File transfer funkció ellenőrzésénél, az ebben levő fájlokat küldjük majd át. Directory listener esetén a from mappát jelöljük ki figyelésre, és majd ide fogunk fájlokat behúzni. A fájlok/mappák, amikkel a teszt során dolgozni fogunk:

```
- a
        - aa
                - ddd.docx
        - cc.jpg
        - dd.docx
        - ee.pdf
- bb
        - aa
                - ddd.docx
- asd
        - Új mappa
        - FORDITANI.txt
- ccc
        - cloud.png
        - FORDITANI.txt
        - transfer.png
- Sok file
        - Új szöveges dokumentum másolata (0).txt
                        . . .
        - Új szöveges dokumentum másolata (29).txt
- b.txt
- c.jpg
- d.docx
- e.pdf
- q.mp3
```

Tesztelt funkció:	File transfer	
Eszközök:	Azonos és eltérő platformok használata mellett	
Fájlok, amik a from mappában találhatóak	Elvárás	Teljesült
b.txt, c.jpg, d.docx, e.pdf, q.mp3	A fájlok átmennek, a módosítási dátumok nem változnak	Igen
b.txt, c.jpg	A fájlok már megvannak a to könyvtárban, semmit se kell velük kezdeni	Igen
b.txt, c.jpg	A b.txt módosításra került, így át kell küldeni, a c.jpg pedig ugyanaz, semmit se kell vele kezdeni	Igen
„bb” mappa	A /bb/aa/ddd.docx megjelenik, rendes módosítási dátummal, almappákat rendesen létrehozzuk	Igen
„a” mappa	Az „a” mappa teljes tartalma átmegy, a fájlok módosítási dátuma nem változik	Igen
egy üres „Új mappa”	Üres mappát nem küldünk át	Igen
„asd” és „ccc” mappa	Az /asd/FORDITANI.txt, és a „ccc” mappában levő 3 fájl átmegy, módosítási dátumuk nem változik	Igen
„Sok file” mappa	Az összes fájl átmegy, a módosítási dátumok nem változnak	Igen
A „from” könyvtár teljes tartalma	Az összes fájl átmegy, a módosítási dátumok nem változnak	Igen

Tesztelt funkció:	Directory listener		
Eszközök:	Azonos és eltérő platformok használata mellett		
Fájlok, amik a figyelt mappába kerülnek	Elvárás	Teljesült	Check failed files után teljesült
b.txt	Létrehozunk egy új dokumentumot (átküldődik), majd átnevezünk b.txt-nek. (átküldődik). 2 fájl kell átküldődjön.	Igen	-
b.txt	Módosítjuk az előbb létrehozott fájlt, a változás megjelenik	Igen	-
b.txt	Bemásoljuk az eredeti b.txt-t. Mivel a „to” mappában az előbb módosított van, nem küldi át	Igen	-
b.txt, c.jpg, d.docx, e.pdf, q.mp3	A fájlok átmennek, a módosítási dátumok nem változnak	Igen	-
„bb” mappa	A /bb/aa/ddd.docx megjelenik, rendes módosítási dátummal, almappákat rendesen létrehozzuk	Igen	-
„a” mappa	Az „a” mappa teljes tartalma átmegy, a fájlok módosítási dátuma nem változik	Igen	-

egy üres „Új mappa”	Létrehozunk egy üres „Új mappa” nevű mappát (átküldődik), majd átnevezünk „valami”-nek. (átküldődik). 2 mappa kell átküldődjön.	Igen	-
e.pdf, q.mp3 a valami mappába	Két fájl behúzunk a frissen létrehozott mappába. A fájlok átmennek, a módosítási dátumok nem változnak	Igen	-
„asd” és „ccc” mappa	Az „asd” és „ccc” mappa tartalma átmegy, módosítási dátumok nem változnak	Igen	-
„Sok file” mappa	Az összes fájl átmegy, a módosítási dátumok nem változnak	Igen	-
A „from” könyvtár teljes tartalma	Az összes fájl átmegy, a módosítási dátumok nem változnak	Igen	-

A tesztek során minden fájl átküldésre került, így nem volt szükség a sikertelen fájlok ellenőrzésére.

7. Összegzés

Az elkészült alkalmazás lehetővé teszi egy kijelölt mappában lévő fájlok átküldését egy másik eszközre, fizikai összeköttetés nélkül. Ezáltal képesek lehetünk kiváltani a telefonunkkal, tabletünkkel, vagy az ezekbe behelyezett SD kártyával a pendrive-ot. Ezt egyszerűen kivitelezhetjük, a számítógépről átküldjük a fájlokat a telefonra, majd később, a telefonról átküldjük a másik számítógépre. Amennyiben egy adott mappában dolgozunk, fájlokat módosítunk, másolunk, lehetőségünk van csak a módosított fájlok szinkronizálására is, ehhez csak az adott mappa figyelését kell kiválasztanunk.

A program elkészítése során fontos szempontok voltak, hogy egy átlagember is egyszerűen és könnyedén el tudjon igazodni és tudja használni, ennek elősegítéséhez az alkalmazásban megfelelő segítséget is találhatunk. A könnyed használatot az is elősegíti, hogy a szövegek változnak, a gombok aktiválódnak és deaktiválódnak a módok változtatásakor, ezáltal szinte eltéveszthetetlené teszi a helyes használatot. Az alkalmazás a Windowson és az Androidos verzióban is egy modern, egyszerű és szép felhasználói felületet kapott ezáltal javítva a felhasználói élményt.

A forráskód jól tagolt, beszélő változónevekkel ellátott, jól kommentezett, így könnyedén szerkeszthető és adható hozzá új funkció, ezáltal lehetőséget adva a továbbfejlesztésre.

Néhány továbbfejlesztési lehetőség:

- Az alkalmazás több operációs rendszerre való portolása, például macOS, Linux
- Ne csak az alhálózaton lévő eszközökre lehessen szinkronizálni
- Kriptográfia használatával a kommunikációt titkosított csatornákon megvalósítani
- Az alkalmazás felhasználói felületének fejlesztése, például különböző nyelvi csomagok hozzáadása, időseknek a gombok, szövegek nagyobb betűméretre való állítása, stílus, hátterek változtatása
- Megtámogatni egy virtuális felhő alapú tároló szolgáltatással, ezáltal a OneDrive-hoz hasonló plusz működést is el lehetne érni
- Megtámogatni a fájlok dátum szerinti verziókezelésével

8. Irodalomjegyzék

[1] #1.How to create/design modern/flat ui using javaFX (JavaFX) [Online] (elérés dátuma: 2021.03.29) https://www.youtube.com/watch?v=T8qBSeJei0U&list=LL&index=4&t=842s

[2] Java: List Files in a Directory [Online] (elérés dátuma: 2021.02.11) https://stackabuse.com/java-list-files-in-a-directory/

[3] Transferring file name then file data over socket [Online] (elérés dátuma: 2021.02.15) https://coderanch.com/t/556838/java/Transferring-file-file-data-socket

[4] Selecting Folder Destination in Java? [Online] (elérés dátuma: 2021.03.03) https://stackoverflow.com/questions/10083447/selecting-folder-destination-in-java/43138765

[5] Watching a Directory for Changes [Online] (elérés dátuma: 2021.03.19) https://docs.oracle.com/javase/tutorial/essential/io/notification.html

[6] Programozási technológia 2. Szálkezelés – 1. [Online] (elérés dátuma: 2021.04.02) http://swap.web.elte.hu/2018192_pt2/ea08.pdf

[7] How to Create Welcome Screen (Splash Screen) in Android Studio [Online] (elérés dátuma: 2021.04.15) https://www.youtube.com/watch?v=jXtof6OUtcE

[8] Create and Use Gradients in Android Studio [Online] (elérés dátuma: 2021.04.15) https://www.youtube.com/watch?v=v8fDkKB-Vz0

[9] gradients.rar/gradient collection/gradient_4.xml [Online] (elérés dátuma: 2021.04.15) https://drive.google.com/file/d/1eQzp_khg3Xq0qTtCKqusOzuHgn9KiV1q/view

[10] Options Menu with Sub Items - Android Studio Tutorial [Online] (elérés dátuma: 2021.04.17) https://www.youtube.com/watch?v=oh4YOj9VkVE

[11] Simple Android Directory picker - How? [Online] (elérés dátuma: 2021.04.18) https://stackoverflow.com/questions/27898676/simple-android-directory-picker-how

[12] How to call a method after a delay in Android [Online] (elérés dátuma: 2021.04.15) https://stackoverflow.com/questions/3072173/how-to-call-a-method-after-a-delay-in-android

[13] UDP Multicast on Android [Online] (elérés dátuma: 2021.04.10) https://codeisland.org/2012/udp-multicast-on-android/

[14] Android 5.0 DocumentFile from tree URI [Online] (elérés dátuma: 2021.04.18) https://stackoverflow.com/questions/34927748/android-5-0-documentfile-from-tree-uri/36162691#36162691

[15] Android: How to list all the files in a directory in to an array? [Online] (elérés dátuma: 2021.04.19) https://stackoverflow.com/questions/27996591/android-how-to-list-all-the-files-in-a-directory-in-to-an-array/27996609

[16] How to Request Permissions in Android Application? [Online] (elérés dátuma: 2021.04.23) https://www.geeksforgeeks.org/android-how-to-request-permissions-in-android-application/?fbclid=IwAR2hJBm3q6wDowFn3Ae2YYu19SDa6u4syWp7eiL1qFoBLM0NJOP8hPKdXwE