



MATEMATIKAI ÉS INFORMATIKAI INTÉZET

# Kliens-szerver kommunikáció Android platformon

**Készítette**

Balajti-Tóth Kristóf

Programtervező Informatikus BSc

**Témavezető**

Tajti Tibor

Egyetemi adjunktus

EGER, 2019

# Tartalomjegyzék

<b>1. Fejlesztői eszközök</b>	<b>5</b>
1.1. Fejlesztői környezetek . . . . .	5
1.1.1. Android Studio . . . . .	5
1.1.2. Pycharm Professional Edition . . . . .	5
1.2. Postman . . . . .	5
<b>2. Platformok</b>	<b>6</b>
2.1. A szerver kiválasztása és felépítése . . . . .	6
2.2. Mobil platform választása . . . . .	7
<b>3. Felhasznált technológiák</b>	<b>8</b>
3.1. Folyamatos integrálás . . . . .	8
3.2. Verzió kezelés . . . . .	8
3.3. Szerveren használt technológiák . . . . .	9
3.3.1. jd-cmd . . . . .	9
3.3.2. Apktool . . . . .	10
3.3.3. dex2jar . . . . .	10
3.3.4. JSON . . . . .	10
3.4. Androidon használt technológiák . . . . .	10
3.4.1. AndroidX . . . . .	10
3.4.2. OkHttp3 . . . . .	11
3.4.3. Okio . . . . .	11
3.4.4. Pusher Beams . . . . .	11
3.4.5. Firebase Messaging . . . . .	11
3.4.6. Room . . . . .	11
3.4.7. CodeView . . . . .	11
3.4.8. Espresso . . . . .	11
3.4.9. Material Design 2.0 . . . . .	11
<b>4. Teszteléshez felhasznált eszközök és források</b>	<b>13</b>
4.1. Sérülékeny Android alkalmazások . . . . .	13

<b>5. Megvalósított funkciók</b>	<b>14</b>
5.1. Bejelentkezés . . . . .	14
5.2. Regisztráció . . . . .	14
5.3. Kijelentkezés . . . . .	14
5.4. Adatok törlése . . . . .	14
5.5. Fájl feltöltés . . . . .	14
5.6. Fájl letöltés . . . . .	14
5.7. Navigáció a fájlrendszerben . . . . .	14
5.8. Értesítések . . . . .	15
<b>6. Továbbfejlesztési lehetőségek</b>	<b>16</b>
<b>7. Tapasztalatok</b>	<b>17</b>

# Bevezetés

Az szoftver fejlesztés egy nagyon komplex folyamat és rengeteg részletre oda kell figyelni. Az elkészült programnak hatékonynak, hibamentesnek és gyorsnak kell lennie. Természetesen, mindezt határidőn belül kell teljesíteni. Sajnos a biztonság nem egy első számú szempont egy megrendelő szemében, csak akkor ha már valami baj történt. Inkább a gyorsaságon és a folyamatok automatizálásán van a hangsúly, ezért nem meglepő, hogy a fejlesztés életciklusának tervezési szakaszában kevés figyelem fordul a szoftver biztonságossá tételére.

A statista.com [1] kutatása szerint 2020-ra több mint 4.78 billió telefon lesz használatban. Ezzel a cégek is tisztában vannak és tudják, hogy ha még több emberhez szeretnék eljuttatni a szolgáltatásukat, akkor rendelkezniük kell saját mobilos alkalmazással.

A mobilos eszközöket célzó támadások száma hatalmas ütemben nő. Mindez azért lehetséges, mert figyelmen kívül marad a „secure coding”-nak nevezett gyakorlat. Egy alkalmazásnak a sebezhetőségét különböző támadási vektoron is ki lehet aknázni. Az elején, bennem többek között az a kérdés merült fel, hogy honnan tudható hogy ez alkalmazás ebezhető-e vagy sem. egy kérdés merült fel. Honnann tudhatom, hogy egy adott alkalmazás sebezhető-e vagy sem. A leghatékonyabb módszer ha visszafejtjük a fájl forráskódra. Ezt angolul „reverse engineering”-nek nevezik. A visszaállított fájlok olvashatósága nem lesz tökéletes, főleg ha obfuszkált <sup>1</sup> kóddal állunk szemben, de egy tapasztalt szem így is kitudja szűrni a gyakori hibákat.

A szakdolgozatomban Android platformra készült telepítő fájlok forrás fájlokká való visszaállításáról írok, valamint bemutatom hogyan valósítható meg a kliens-szerver kommunikáció egy REST API és egy Androidos alkalmazás segítségével. A projektet „Reverse Droid”-nak neveztem el.

---

<sup>1</sup> Az obfuszkáció célja röviden, hogy megnehezítse a visszafejtett kód olvashatóságát.

# 1. fejezet

## Fejlesztői eszközök

### 1.1. Fejlesztői környezetek

#### 1.1.1. Android Studio

Az Android Studio jelenleg az egyetlen jól támogatott és minőségi fejlesztői környezet Android fejlesztéshez. Régebben sok panaszt hallottam az emulátorára, hogy nagyon lassú és körülményes a használata. Mára már egy pillanat alatt lehet futtatni a programunk és abszolút kényelmes lett a használata. Rendelkezik APK elemzővel, vizuális felhasználó felület szerkesztővel és intelligens kód szerkesztővel is. Az egyik kedvenc funkcióm a valós idejű profilozó, ami segítségével megtudjuk nézni valós időben, milyen erőforrásokat használ az alkalmazásuk. Ez különösen hasznos, ha megakarunk találni egy memória szivárgást vagy egy olyan részt, ami a kelleténél jobban meríti az akkumulátorunk. Említésre méltó még a flexibilis build rendszere is, a Gradle. Használatával megtehetjük, hogy külön build típusokat hozunk létre a különböző eszközökre. [2]

#### 1.1.2. Pycharm Professional Edition

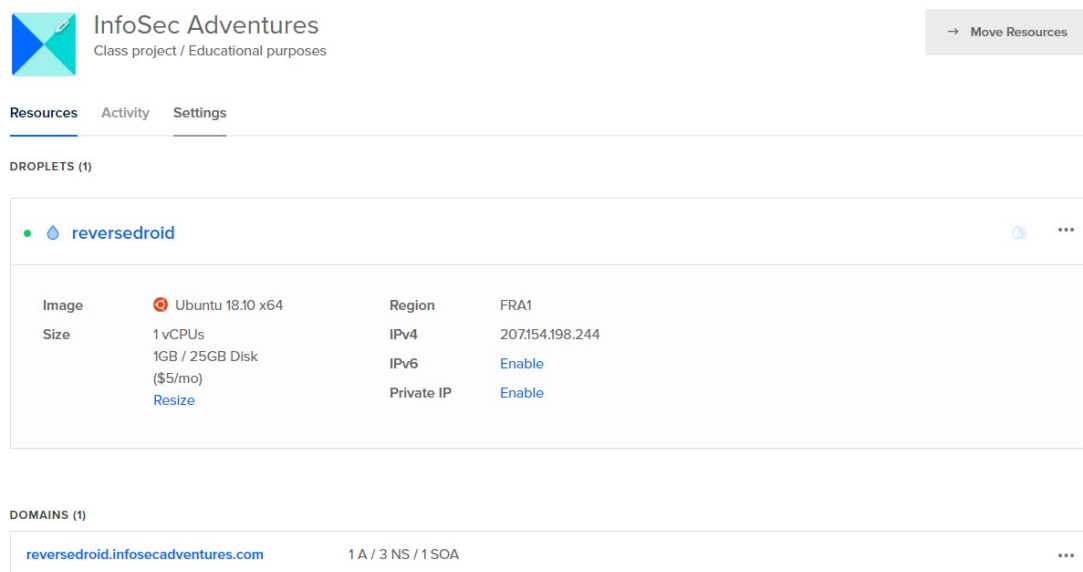
### 1.2. Postman

## 2. fejezet

# Platformok

### 2.1. A szerver kiválasztása és felépítése

Olyan szerverre volt szükségem, ami nem túl költséges, de mégis megfelelően testreszabható és gyors tárhelyet biztosít. A választásom a Digital Ocean felhő szolgáltatására esett. Az oldal felületén lehetőségünk van több, úgynevezett *droplet*-et létrehozni, amik nem mások mint virtuális szerverek. Megadhatjuk milyen disztribúciót szeretnénk telepíteni, jelen esetben én az Ubuntu Linux 18.10-es verzióját telepítettem.



2.1. ábra. Droplet a Digital Ocean admin felületén.

A projecthez készítettem egy subdomain-t és telepítés után a droplet IP címét hozzárendeltem ehhez a subdomain-hez. Ezzel biztosítottam, hogy domain név alapján is elérhető legyen a szerver. Ez a 2.2 képen jól látható.

A kész projectben nem ezt a folyamatot választottam, hanem a Digital Ocean által nyújtott „one-click apps” menüben egyszerűen kiválasztottam a Docker alkalmazást és

Type	Host	Value	TTL
A Record	@	185.199.108.153	Automatic
A Record	@	185.199.109.153	Automatic
A Record	@	185.199.110.153	Automatic
A Record	link	52.72.49.79	Automatic
A Record	reversedroid	207.154.198.244	Automatic

2.2. ábra. DNS rekordok a domain beállításában.

az elkészült képfájlt ezen futattam. Így automatizálva a szerver telepítésének folyamatát és megspórolva magának a Docker-nek a telepítését és konfigurálását. Erről még a Szerveren használt technológiák fejezet Docker alfejezetében bővebben írok.

## 2.2. Mobil platform választása

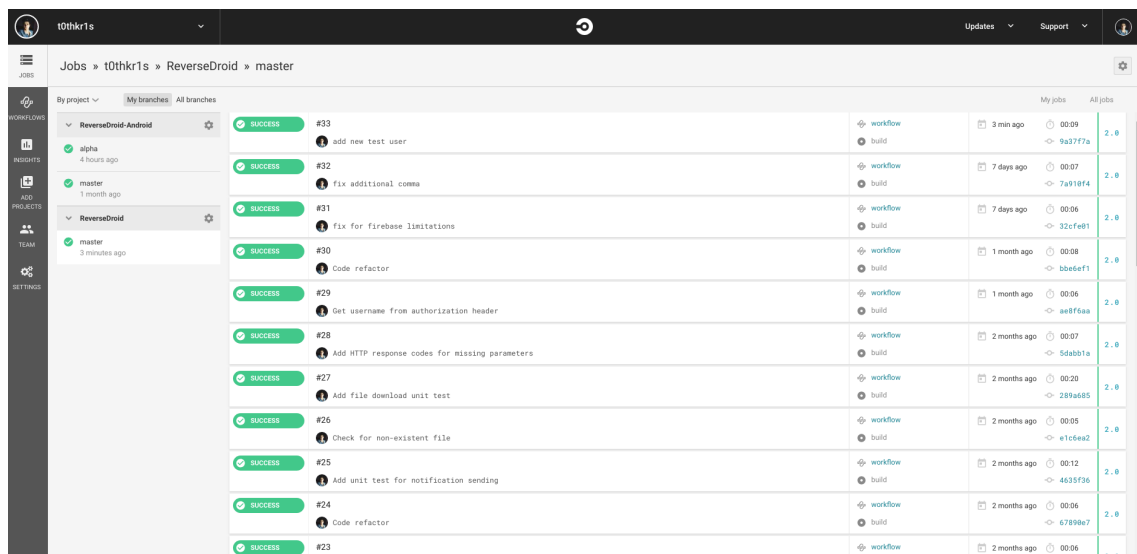
A mobilos operációs rendszerek közül az Androidot választottam. Már korábban sikerült megismerkednem az Android nyújtotta lehetőségekkel és előnyökkel. A többi mobilos operációs rendszerrel ellentétben az Android nyílt forráskódú és a piac több mint felét uralja. Ez annak is köszönhető, hogy 2005-ben a Google felvásárolta az Android projectet és azóta ők tartják karban. A fejlesztő környezete elérhető mind a három fő operációs rendszerre (Linux, macOS, Windows). Számomra ezek voltak a legnyomósabb érvek a rendszer kiválasztásában.

# 3. fejezet

## Felhasznált technológiák

### 3.1. Folyamatos integrálás

A folyamatos integrálás egy extrém programozási gyakorlat. A folyamatos integrálás arról szól, hogy ha egy feladat elkészült akkor azt egyből beintegráljuk a rendszerbe. A beintegrálás után természetesen minden egység tesztnek sikeresen le kell futnia. Több nagy cég is a *CircleCi*-t használja a folyamatos integráláshoz. Ilyen például a *Facebook*, *Spotify*, *Kickstarter* és a *GoPro*.



Jobs » t0thkr1s » ReverseDroid » master		My jobs		All jobs	
By project	My branches	All branches			
ReverseDroid-Android	alpha	4 hours ago	success	#33	add new test user
	master	1 month ago	success	#32	fix additional comma
ReverseDroid	master	3 minutes ago	success	#31	fix for firebase limitations
			success	#30	Code refactor
			success	#29	Get username from authorization header
			success	#28	Add HTTP response codes for missing parameters
			success	#27	Add file download unit test
			success	#26	Check for non-existent file
			success	#25	Add unit test for notification sending
			success	#24	Code refactor
			success	#23	

3.1. ábra. Sikeres project buildek a CircleCi felületén.

### 3.2. Verzió kezelés

Már a project kezdetekor készítettem egy privát Github repository-t, hogy nyomon tudjam követni a változtatásaimat és esetleges hiba esetén visszaállítani egy korábbi verzióra.



## 3.3. Szerveren használt technológiák

### Flask

A szervert a Flask webes mikro keretrendszer felhasználásával készítettem el. A *LinkedIn* és *Pinteres* is említésre méltó alkalmazások, amik felhasználták ezt a keretrendszert.

### SQLite

Az SQLite a legtöbbet használt adatbázis motor a világon. Számtalan alkalmazás használja és Android-on is ez az alapértelmezett adatbázis. Az SQLite nyílt forráskódú, így mindenki nyugodtan használhatja.

### Pusher Beams

A szerverről való értesítés küldéshez a Pusher Beams SDK-ját használtam. A Beams SDK lehetővé teszi, hogy egyszerűen küldjünk push értesítést „érdeklődési” körök alapján és platformtól függetlenül. Ingyenesen és korlátlanul küldhetjük ezeket az értesítéseket. A szerver és kliens is oldal implementációja is nagyon egyszerű és jól dokumentált, de ha mégis problémánk támadna a support is végtelenül segítőkész.

### Docker

A szerverhez készítettem egy Dockerfile-t és csatoltam a project Github-os repository-ját. Ezzel elérve, hogy minden egyes változtatásnál a Docker Hub újra buildelje a képfájlt. A folyamat nagyon hasonlít a folyamatos integrálásra.

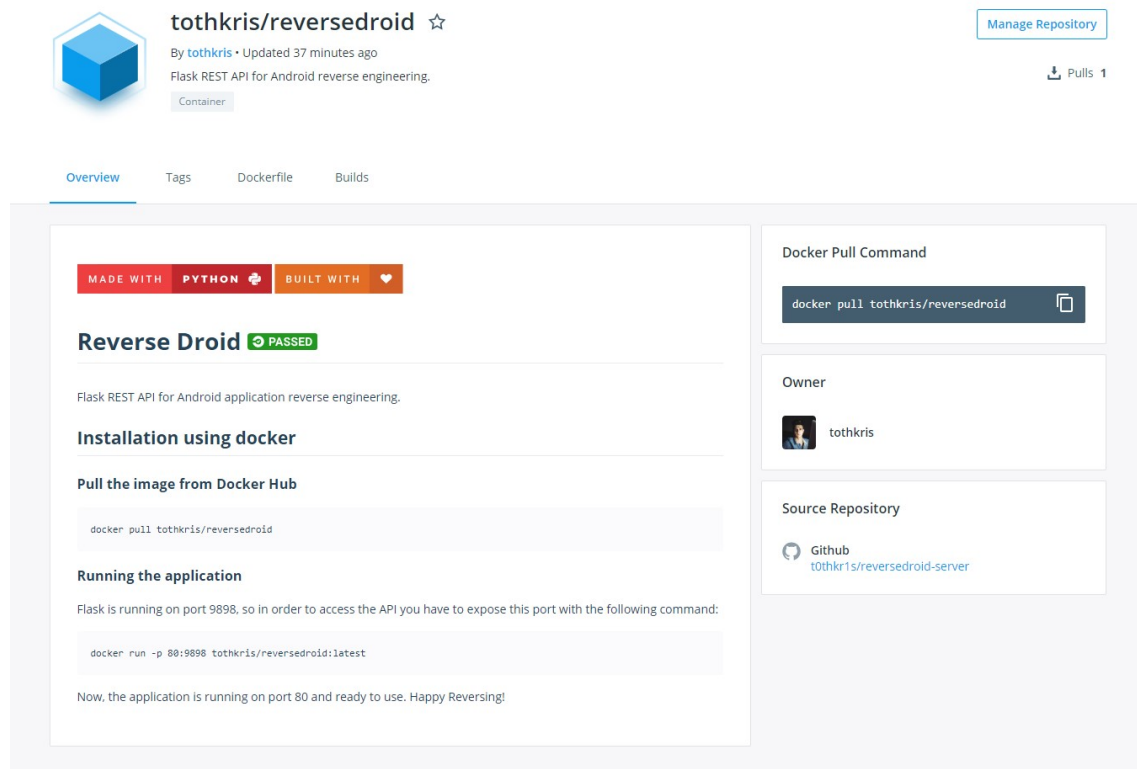
### Tmux

A Tmux vagy másnéven terminál multiplexer használatával több shell-t lehet futtatni. Ez még nem is lenne túl nagy segítség, viszont ezt a folyamatot le lehet csatolni majd vissza anélkül, hogy elveszne a tartalom. Ezen felül biztosítja, hogy a parancsok és azok kimenete kereshetők legyenek. Az alkalmazás alapvetően a parancssorba írja a bejövő kéréseket és ez egy idő után nehezen követhető. A Tmux nagy segítségemre volt, mivel lehetőséget adott hogy keressek a kérések között.

#### 3.3.1. jd-cmd

Szükségem volt egy parancssorból használható Java Decompiler-re is.

<https://github.com/kwart/jd-cmd>



3.2. ábra. A szerver Docker Hub-on is elérhető.

### 3.3.2. Apktool

<https://github.com/iBotPeaches/Apktool>

### 3.3.3. dex2jar

Android .dex és .class fájlokkal való dolgozáskor kulcs szerepet játszott a *dex2jar*. Jelen esetben nem csak egy programról beszélünk, mert valójában több eszközt foglal magába. Az Android telepítő fájl kitömörítése után, az alkalmazás forráskódja egy *classes.dex* fájlban található meg. Ez a fájl byte kódot tartalmaz az ART számára és ezt kellett átalakítani .jar formátumba, amivel már a decompiler dolgozni tud.

<https://bitbucket.org/pxb1988/dex2jar>

### 3.3.4. JSON

## 3.4. Androidon használt technológiák

### 3.4.1. AndroidX

Az AndroidX egy nyílt forráskódú projekt, amit az Android fejlesztői csapata használ library-k fejlesztéshez, teszteléséhez és kiadásához a Jetpack-en belül. Az eredeti sup-

port library-hez képest az AndroidX egy jelentős fejlődés. Ahogy a support library-t is, az AndroidX-et is az Android operációs rendszertől függetlenül tudjuk használni és biztosítja a visszafelé kompatibilitást. Az AndroidX teljesen felváltja a support library-t azáltal, hogy azonos funkciókat biztosít és új könyvtárakat. Ezen felül az AndroidX a következő funkciókat tartalmazza:

1. Minden csomag egy konzisztens névtérben van, ami „androidx”-el kezdődik.
2. A support library-vel ellentétben az AndroidX csomagok külön vannak karbantartva és frissítve.
3. Az összes új support library fejlesztés az AndroidX könyvtárban fog történni. Ez magába foglalja az eredeti support library fenntartását és az új Jetpack komponensek bevezetését.

### **3.4.2. OkHttp3**

### **3.4.3. Okio**

### **3.4.4. Pusher Beams**

### **3.4.5. Firebase Messaging**

### **3.4.6. Room**

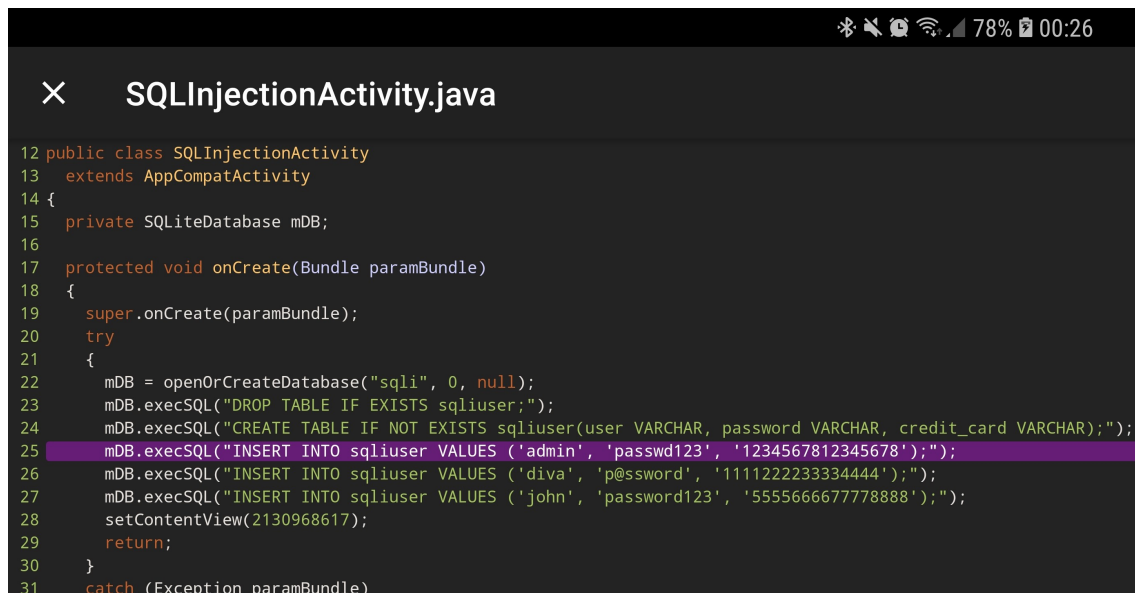
A Room egy absztrakciós réteget nyújt az SQLite adatbázishoz, ezzel lehetővé téve az egyszerűbb és hatékonyabb adatbázis elérést. A Room segítségével sokkal egyszerűbben tudtam kezelni az SQLite adatbázist, mert nem kellett adatbázist leíró osztályt és hosszú lekérdezéseket írnom. Ami külön tetszett benne, hogy az SQL utasításokat fordítási időben ellenőrzi. Annotációk segítségével tudjuk összekötni a Java POJO osztályokat az SQLite adatbázissal.

### **3.4.7. CodeView**

A forrás kód megjelenítését nem lett volna célszerű nulláról felépíteni, ezért inkább kész megoldások után néztem. Több megfelelő forrás kód megjelenítő könyvtárat találtam, így funkciók alapján kellett döntenem, hogy melyiket válasszam.

### **3.4.8. Espresso**

### **3.4.9. Material Design 2.0**



```
12 public class SQLInjectionActivity
13     extends AppCompatActivity
14 {
15     private SQLiteDatabase mDB;
16
17     protected void onCreate(Bundle paramBundle)
18     {
19         super.onCreate(paramBundle);
20         try
21         {
22             mDB = openOrCreateDatabase("sqli", 0, null);
23             mDB.execSQL("DROP TABLE IF EXISTS sqliuser;");
24             mDB.execSQL("CREATE TABLE IF NOT EXISTS sqliuser(user VARCHAR, password VARCHAR, credit_card VARCHAR);");
25             mDB.execSQL("INSERT INTO sqliuser VALUES ('admin', 'passwd123', '1234567812345678');");
26             mDB.execSQL("INSERT INTO sqliuser VALUES ('diva', 'p@ssword', '1111222233334444');");
27             mDB.execSQL("INSERT INTO sqliuser VALUES ('john', 'password123', '5555666677778888');");
28             setContentView(2130968617);
29             return;
30         }
31         catch (Exception paramBundle)
```

3.3. ábra. A vissza fejtett forrás kód megjelenítése.

## 4. fejezet

# Teszteléshez felhasznált eszközök és források

### 4.1. Sérülékeny Android alkalmazások

A teszteléshez keresnem kellett olyan sérülékeny alkalmazásokat, amelyeken jól lehet demonstrálni a visszafejtést. Minden alkalmazás, amit itt megemlítek nyílt forrás kódú és kimondottan erre a célra készítették őket. Ez azért is jó, mert a visszafejtett kódot össze lehet hasonlítani az eredetivel és megnézni mennyire volt hatékony a visszafejtési folyamat. A következő lista tartalmazza a teszteléshez felhasznált szándékosan sérülékeny alkalmazásokat.

- Diva
- Sieve
- Pivaa
- Frida

## 5. fejezet

# Megvalósított funkciók

### 5.1. Bejelentkezés

### 5.2. Regisztráció

### 5.3. Kijelentkezés

### 5.4. Adatok törlése

### 5.5. Fájl feltöltés

### 5.6. Fájl letöltés

### 5.7. Navigáció a fájlrendszerben

A letöltött és kitömörített projektek az alkalmazás mappájában kerülnek. Ezzel alapvetően nincs is semmi probléma, csak ha megszeretnénk nézni a fájlokat, akkor kellene hagynunk az alkalmazást. Ez pedig rossz felhasználói élményhez vezetett volna. Ennek megoldására megvalósítottam egy nagyon egyszerű fájl rendszer navigációt, de ezt csak az alkalmazás mappájában tettem lehetővé. Így kitömörítés után egyszerűen és gyorsan elérhetővé válik a project a felhasználó számára. Részleteit tekintve csak a fájl nevét, módosítás dátumát és a fájl méretét jelenítettem meg.

## 5.8. Értesítések

Az értesítések küldése és fogadása szintén egy kulcs fontosságú része az alkalmazásnak. A fájlok letöltése és kitömörítése mérettől, valamint hálózati kapcsolattól függően időt vesz igénybe.

A szerveren történő visszafejtés az a folyamat, ami jelentős időtartamot igényel.

## 6. fejezet

# Továbbfejlesztési lehetőségek

Úgy gondolom, hogy sokkal nagyobb piaci érték rejlik ebben az alkalmazásban. A jövőben is szeretném folytatni a fejlesztést. Szeretnék több figyelmet fordítani a biztonságra és hatékonyságra. Gondolok itt a biztonságos kommunikációra TLS-es keresztül és a harmadik féltől származó könyvtárak csökkentésére. A kód visszafejtése végén egy összegző report is hasznos lehet a felhasználó számára, ami tartalmazhatja a feldolgozott fájlok számát. Egy esetleges forrás kód elemző



## 7. fejezet

# Tapasztalatok

Úgy érzem elértem a célokat ezzel a projecttel és sokat sikerült tanulnom a folyamatban. Új technológiákat ismertem meg és használtam. Természetesen nem volt minden magától értetődő és jó pár nehézséggel is találkoztam.

# Irodalomjegyzék

- [1] ONLINE: Number of mobile phone users worldwide from 2015 to 2020, <https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide>
- [2] ONLINE: Everything you need to build on Android <https://developer.android.com/studio/features.html>