



MATEMATIKAI ÉS INFORMATIKAI INTÉZET

Kliens-szerver kommunikáció Android platformon

Készítette

Balajti-Tóth Kristóf

Programtervező Informatikus BSc

Témavezető

Tajti Tibor

Tanársegéd

EGER, 2019

Tartalomjegyzék

1. Fejlesztői környezetek	8
1.1. Android Studio	8
1.2. Pycharm Professional Edition	8
1.3. Postman	9
2. Platformok	10
2.1. A szerver kiválasztása és felépítése	10
2.2. Mobil platform választása	10
3. Felhasznált technológiák	12
3.1. Verzió kezelés	12
3.2. Folyamatos integrálás	13
3.3. Szerveren használt technológiák	14
3.3.1. Flask	14
3.3.2. SQLite	14
3.3.3. Pusher Beams (Python Server SDK)	15
3.3.4. Docker	16
3.3.5. Apktool	17
3.3.6. dex2jar	18
3.3.7. jd-cmd	18
3.3.8. JSON	19
3.4. Androidon használt technológiák	20
3.4.1. AndroidX	20
3.4.2. OkHttp	20
3.4.3. Pusher Beams (Java Client SDK)	21
3.4.4. Room	21
3.4.5. CodeView	22
3.4.6. Espresso	25
3.4.7. Material Design 2.0	27
4. A visszafejtés teszteléséhez felhasznált programok	28
4.1. Sérülékeny Android alkalmazások	28

4.1.1.	Célközönség	29
4.1.2.	Sérülénységek	29
5.	Megvalósított funkciók	30
5.1.	Regisztráció	30
5.2.	Bejelentkezés	30
5.3.	Kijelentkezés	31
5.4.	Adatok törlése	32
5.5.	Fájl feltöltés	33
5.6.	Fájl letöltés	33
5.7.	Navigáció a fájlrendszerben	34
5.8.	Értesítések	35
6.	Továbbfejlesztési lehetőségek	37
7.	Tapasztalatok	38

Ábrák jegyzéke

3.1.	Project buildek követése a CircleCi felületén.	13
3.2.	A publikált applikáció Docker Hub-on.	18
3.3.	A visszafejtett forráskód megjelenítése szintaktikai kijelöléssel.	24
5.1.	A bejelentkezési és regisztrációs felület megvalósítása.	31
5.2.	A feltöltött APK fájlok és értesítések felülete.	34
5.3.	Az alkalmazásban megjelenő értesítések.	36

Forráskódok jegyzéke

1.	Egy nagyon egyszerű példa a szerverről érkező JSON válaszra.	19
2.	A NotificationDao interfész.	23
3.	A CodeView könyvtár felhasználása és kiegészítése.	24
4.	Espresso UI teszt az adatok törléséhez.	26
5.	Adatok mentése <i>SharedPreferences</i> segítségével.	32
6.	Az összes <i>SharedPreferences</i> -be mentett adat törlése.	32

Bevezetés

A szoftver fejlesztés egy nagyon komplex folyamat és rengeteg részletre oda kell figyelni. Az elkészült programnak hatékornynak, 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 applikációval. A kérdés az, hogy ezt a hatalmas keresletet mennyire tudják a fejlesztők kielégíteni és még ha ki is tudják, milyen minőségű kódot tudnak előállítani. A fejlesztőkkel szemben nagyon sok elvárás támasztanak a megrendelők és ahogy előbb is említettem a biztonság csak ritkán elvárás. Véleményem szerint ha nem szentelnek elég figyelmet a behatolás tesztelésre és magára a biztonságra általánosságban, akkor annak elég súlyos következményei lehetnek.

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 egy alkalmazás sebezhető-e vagy sem. Honnann tudhatom, hogy egy adott alkalmazás sebezhető-e vagy sem és ha igen milyen súlyos a hiba. A leghatékonyabb módszer ha megpróbáljuk visszafejteni az alkalmazást forráskódra. Ezt angolul „reverse engineering”-nek nevezik. A visszaállított fájlok olvashatósága nem lesz tökéletes, főleg ha obfuszkált ¹ kóddal állunk szemben, de egy tapasztalt szem így is kitudja szűrni a gyakori hibákat.

A szakdolgozatomban Android platformra készült alkalmazások forrás fájlkká való visszaállításáról írok, valamint bemutatom hogyan valósítható meg a kliens-szerver kommunikáció egy Python programozási nyelvben írt REST² API és egy Android platformra készült kliens segítségével. Szó lesz arról, hogy hogyan tudunk fájlokat feltölteni,

¹ Az obfuszkáció célja röviden, hogy megnehezítse a visszafejtett kód olvashatóságát.

² Representational State Transfer

valamint letölteni a szerverről és értesítésekkel operálni. A felhasználó egy egyszerű autentikáció után képes lesz *.apk* fájlok feltöltésére, letöltésére és az elkészült projektben való navigálásra. Hosszabb ideig tartó folyamatok állapotáról és elkészültéről értesítést kap a felhasználó és lehetősége lesz a forráskód alkalmazáson belüli megtekintésére és megosztására. Ezzel egy hatékony és sokoldalú eszközt adva a tesztelők, kutatók és a terület után érdeklődők kezébe. A projektet „Reverse Droid”-nak neveztem el.

1. fejezet

Fejlesztői környezetek

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. Az emulátor állapota menthető, ezáltal indításkor ott folytathatjuk ahol abba hagytuk. Azon kívül, hogy használatával több különböző eszközön tesztelhetjük az alkalmazásunk, lehetőséget ad a szenzorok, hálózati és GPS kapcsolat szimulálására. 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 kellénél 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. Az *instant run* funkció segítségével egyből tudjuk futtatni a kódban véghez vitt kisebb változtatásokat, anélkül hogy újraindítanánk az Activity -t vagy újra buildelnénk az egész projektet és új APK -t telepítenénk. [2]

1.2. Pycharm Professional Edition

A PyCharm is egy IDE ¹, mint az Android Studio. Dolgozhatunk webes technológiákkal vagy mesterséges intelligenciával, a PyCharm megfelelő választás lehet bármilyen területen programozó számára. A Pycharm mögött is a *Jetbrains* cég áll. Ezt azért fontos megemlíteni, mert már 15 éve azon dolgoznak, hogy a legjobb és leghatékonyabb fejlesztői környezetek állítsanak elő. Véleményem szerint ez sikerült is nekik. Az Android

¹ Integrated Development Environment (integrált fejlesztői környezet)

Studio-n és a Pycharm-on is látszik, hogy minőségi termékek és rengeteget segítenek a fejlesztők mindennapjaiban. Én a *PyCharm Professional Edition*-t használtam, amihez a diákok ingyenesen hozzájuthatnak. Mivel adatbázissal is dolgoznak, ezért a *Community Edition* nem lett volna megfelelő. Nem csak az adatbázis támogatást nyújt, hanem webes keretrendszer támogatást és profilozót is. A távoli fejlesztés funkció is rendkívül praktikus. A fejlesztés közben egyszerűen tudtam feltölteni a szerverre a változtatásaimat. A verzió kezelőknek is egyesített felületet nyújt, amivel jelentős időt spórolhatunk meg.[9] Ez a lehetőség mindkettő fejlesztői környezetben elérhető.

1.3. Postman

Jelenleg a Postman a legnépszerűbb API² tesztelésben használt eszköz. A Postman kollektciók futtatható leírásai egy API-nak és sarok kövei a Postman beépített eszközeinek. Ezeknek a beépített eszközöknek köszönhetően futtathatunk hálózati kéréseket, teszteket, debuggolhatunk és csinálhatunk mock szervereket is. Ráadásul automatizáltan futtathatjuk a teszteket és egyszerűen elkészíthetjük és publikálhatjuk az API dokumentációját. Én kizárólag dokumentáció készítésre és a végpontok tesztelésére használtam. Ettől természetesen sokkal több lehetőség rejlik benne. Segítségével egyszeren publikálhattam az API dokumentációját, így lehetővé téve a felhasználást mások számára is. A Postman a <https://documenter.getpostman.com/view/5052599/S11RKayE> oldalon található leírást készítette el számomra. Különösen hasznos, hogy nem csak a végpontokat mutatja be hanem a kéréshez szükséges fejléceket, paramétereket és még a törzs típusát is. A weboldalon ezeken felül kiválaszthatjuk, hogy milyen programozási nyelvben kívánjuk megírni a saját kliensünk. Miután ezt megtettük, az oldal jobb oldalán megjelennek a kérésekhez tartozó nyelv specifikus példák. Úgy gondolom, ez a funkció hatalmas segítséget nyújt a dokumentációt felhasználók számára.

² Application Programming Interface

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 választottam.

Szerencsére nem kellett új domain nevet vásárolnom, mert már rendelkeztem egyel. A domain egyébként a *Namecheap* oldalon van beregisztrálva. Annyi volt a dolgom, hogy egy új aldomaint kellett készítenem a projekt számára. Miután feltelepült a Droplet hozzárendeltem az IP¹ címét az előbb elkészített aldomainhez. Ezzel biztosítottam, hogy domain név alapján is elérhető legyen a szerver. Körülnéztem a konkurens cégek által nyújtott szolgáltatások között is, de én úgy láttam, hogy egyik se éri el azt az ár-érték arányt, amit a *Namecheap* nyújt.

Habár nagyon egyszerű volt a folyamat, az elkészült projektben mégsem ezt a folyamatot választottam, hanem a Digital Ocean által nyújtott „one-click apps” menüben egyszerűen kiválasztottam a Docker alkalmazást és az elkészült képfájlt ezen futtattam. Így automatizálva a szerver telepítésének folyamatát és megspórolva a Docker 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. Bevallom őszintén, nem volt nehéz a döntés. Android platform rengeteg olyan lehetőséget biztosít, amivel a többi platform nem szállhat versenybe. Itt gondolok az egyszeri és nem megújuló

¹ Internet Protocol

fizetésre a fejlesztői fiókért, valamint az iOS-el ellentétben a fejlesztői környezete elérhető mind a három fő operációs rendszerre (Linux, macOS, Windows). Számomra ezek elég nyomós érvek voltak, ezért döntöttem úgy, hogy Android platformra fog először elkészülni a program. A Windows Phone a Microsoft cégnek volt egy próbálkozása, ami végül kudarcba fulladt, ezért ez az opció nem jöhetett szóba.

A többi mobilos operációs rendszerrel ellentétben az Android nyílt forráskódú és a piac jelentős részét uralja. Engem különösen megfogott az általa nyújtott szabadság és testre szabhatóság. Az iOS ezzel ellentétben arról híres, hogy egyszerű és megbízható a használata. Viszont jelen vannak olyan megkötések, amik mind a végfelhasználókat és mind a fejlesztőket limitálják. Gondolok itt a minimális testre szabhatóságra és a nagy részt zárt forráskódra. Itt most nyilván nem szeretném részletezni, hogy melyik operációs rendszer a jobb vagy éppen rosszabb, mert mindegyik rendszernek megvan az előnye, illetve a hátránya. Inkább csak a személyem érveimet és tapasztalataimat sorakoztatom fel, amik alapján platformot választottam.

Fontos megemlíteni, hogy az Android rendszer nem csak mobil telefonokban terjedt el. Jelen van az okos televíziókban (Android TV), autókban (Android Auto), okos órákban (Wear OS) és IoT²-ben is (Android Things). Ezek az Android által úgymond eredmények annak is köszönhetőek, hogy a Google felvásárolta az Android céget és azóta ők tartják karban.

Természetesen nem csak jó tulajdonságokkal rendelkezik, hanem bizony van néhány hátránya is. Ide tartozik például a gyártóktól függő frissítések. Tegyük fel, hogy új biztonsági rést fedeztek fel az Android operációs rendszerben. A Google általában ezekre meglehetősen gyorsan reagál és néhány napon belül frissítést ad ki az eszközeire (Nexus, Pixel). Azoknak a gyártóknak akiknek saját testre szabott rendszerük van, jóval tovább tart orvosolni a hibát. Ez az időtartam a mai napig hónapokban mérhető, de persze ez függ az adott cégtől és a hiba súlyosságától.

Számomra ezek voltak a legnyomósabb érvek a rendszer kiválasztásában. A mai napig úgy gondolom, hogy ebben a platformban van a legnagyobb potenciál a fejlesztők és felhasználók számára egyaránt.

² Internet of Things

3. fejezet

Felhasznált technológiák

3.1. Verzió kezelés

Egy verzió kezelő rendszer képes kezelni egy fájl vagy akár több fájl módosításait olyan módon, hogy lehetőségünk legyen időben „visszamenni” és megnézni egy fájl bizonyos verzióját. A Linux kernel forráskódja óriási méretű és a Git verzió kezelőt használják a fejlesztéshez. Ez azért érdekes, mert nagyon sok ember dolgozik egy óriási kódbázissal és a Git mégis képes hatékonyan kezelni a változásokat. A alábbi listában látható, hogy miért bíznak meg benne egy ilyen nehéz feladat esetében és hogyan teljesíti a Git minden elvárásukat:

- Atomosság
- Teljesítmény
- Biztonság

Az atomosság biztosítja, hogy az adatok ne vesszenek el és ne történjen verzió eltérés részlegesen befejezett műveletek miatt. A gyorsasága mellett nem használ fel jelentős mennyiségű tárhelyet, ellentétben a többi verzió kezelő rendszerrel. Ha Git-et használunk biztosra mehetünk, hogy senki sem módosítja a fájlok tartalmát. Ez a SHA-1 kivonatolásnak köszönhető.[5]

A fentiek fényében nyilvánvaló, hogy a Git mellett döntöttem, ami széles körben elterjedt a szoftver fejlesztők között. A Git egy ingyenes és nyílt-forráskódú elosztott verzió kezelő rendszer. Úgy készült, hogy gyorsan és hatékonyan tudjon kezelni kis és nagy projekteket is egyaránt. Már a project kezdetekor készítettem egy privát Github repository-t, hogy nyomon tudjam követni a változtatásaimat és esetleges hiba esetén visszaállítani egy korábbi verzióra.

A Github nem összetévesztendő a Git-el, mert a Git a forráskód változtatásainak kezelésére szolgál (lokálisan), a Github pedig egy tárhelyet nyújt a verzió kezelt könyvtárak és fájlok tárolására. A projekt fejlesztése alatt több gépről dolgoztam és Github

a segítségével biztosítani tudtam, hogy mindenhol elérjen a projektek legfrissebb változtatásait. Lényegében egy központi szerverként szolgált számomra.

3.2. Folyamatos integrálás

A folyamatos integrálás egy szoftver fejlesztési gyakorlat, ahol a csapat tagjai sűrűn integrálják a munkájukat általában napi rendszerességgel. Ez naponta több integrációhoz vezet. Pontosabban megfogalmazva a folyamatos integrálás egy extrém programozási gyakorlat. A folyamatos integrálás arról szól, hogy ha egy feladat elkészült akkor azt egyből beintegráljuk a rendszerbe. Minden integráció hitelesítve van egy automatikus build rendszer által (egység tesztekkel együtt) annak érdekében, hogy minél hamarabb észrevegyük az integrációs hibákat. A beintegrálás után természetesen minden egység tesztnek sikeresen le kell futnia. Sok csapat úgy gondolja, hogy ez a megközelítés jelentősen kevesebb integrációs problémához vezet és meggyorsítja a fejlesztés menetét.[3]

Több nagy cég is a CircleCi szolgáltatását használja a folyamatos integráláshoz, ilyen például a *Facebook*, *Spotify* és a *GoPro*. A CircleCi összeköthető a Github-al, így kényelmesen tudjuk csatolni az ott tárolt projektjeinket. A konfigurációs fájl létrehozása pedig végtelenül egyszerű, ha követjük a dokumentációt. A 3.1 képen látható, hogy minden egyes változtatáskor lefutnak a tesztek. Ezek a tesztek minden alkalommal egy tiszta konténerben vagy virtuális gépen futnak. Jelen esetben az Android program egy Docker konténerben fut, aminek a neve *circleci/android:api-28-alpha*. Az eredményről mindig értesítést kapunk, így egyből tudhatjuk azt is, ha egy build nem volt sikeres.

By project	My branches	All branches
reversedroid-android	alpha 22 hours ago	master 3 months ago
reversedroid-server	master 3 days ago	

STATUS	Job Name	Workflow	Build	Time	Link
SUCCESS	t0thkr1s / reversedroid-server / master #43	workflow	build	7 days ago	00:14 6ce8bb8
SUCCESS	t0thkr1s / reversedroid-server / master #42	workflow	build	7 days ago	00:09 c731ee7
SUCCESS	t0thkr1s / reversedroid-android / alpha #100	workflow	build	12 days ago	00:51 1760743
SUCCESS	t0thkr1s / reversedroid-android / alpha #99	workflow	build	14 days ago	01:23 ff14329
SUCCESS	t0thkr1s / reversedroid-android / alpha #98	workflow	build	14 days ago	01:01 e6aa4fb
FAILED	t0thkr1s / reversedroid-android / alpha #97	workflow	build	14 days ago	01:54 f92b2b9
FAILED	t0thkr1s / reversedroid-android / alpha #96	workflow	build	14 days ago	01:01 2582975
FAILED	t0thkr1s / reversedroid-android / alpha #95	workflow	build	14 days ago	01:31 3a31263
SUCCESS	t0thkr1s / reversedroid-android / alpha #94	workflow	build	14 days ago	00:56 d2af1b8

3.1. ábra. Project buildek követése a CircleCi felületén.

3.3. Szerveren használt technológiák

3.3.1. Flask

A szervert a Flask webes mikro keretrendszer felhasználásával készítettem el. Attól, hogy mikro keretrendszer még nem jelenti azt, hogy kevesebb tud mit a többi keretrendszer. Erős alapot nyújt az alapvető szolgáltatásokkal, miközben a kiegészítők nyújtják a többit. Azzal a lehetőséggel, hogy kitudjuk választani magunknak a kiegészítő csomagokat, képesek vagyunk egy olyan programot felépíteni, ami pontosan azt tartalmazza amire szükségünk van és nem tartalmaz semmi fölösleget.

A Flask nem támogatja natívan az adatbázis elérést, web formok validálását, felhasználói autentikációt és egyéb magas szintű feladatokat. Ezek és más egyéb kulcs szolgáltatások, amikre egy webes applikációnak szüksége van kiegészítőként keresztül elérhető. Fejlesztőként lehetőségünk van egyesével kiválogatni vagy éppen magunktól megírni azokat a kiegészítőket, amik az aktuális projekthez kellenek. [6]

Úgy gondolom, hogy sokkal hatékonyabban programot lehet írni, ha csak azt építjük bele a szoftverbe, amire tényleg szükségünk van. Nekem csak kettő kiegészítőre volt szükségem, mivel a többi funkcionalitást alaphoz nyújtja a keretrendszer.

Flask-Testing

Az egyik ilyen a „Flask-Testing” csomag, amire az egység tesztek miatt volt szükségem. Segítségével rendkívül egyszerűen és hatékonyan tudjuk tesztelni az alkalmazásban található végpontokat. Használata számomra nagyon könnyű volt és a dokumentáció minden felhasználási esetre kitért és lehetővé tette, hogy kényelmesen teszteljem a JSON válaszokat. A dokumentáció azt ajánlja, hogy tegyünk minden tesztet egy nagy fájlba, mert így kényelmesen tudjuk egyszerre futtani őket.

Flask-Bcrypt

Mivel alaphoz a Flask keretrendszer nem biztosít titkosítást, keresnem kellett egy külső megoldást. A másik ilyen csomag amit felhasználtam, a „Flask-Bcrypt” volt. Nagyon rossz gyakorlat, ha a jelszavakat titkosítás nélkül tároljuk az adatbázisban. Ennek elkerülése érdekében választottam a bcrypt algoritmust szolgáltató csomagot, ami kifejezetten erre a célra készült.

3.3.2. SQLite

Az SQLite egy nyílt forráskódú szoftver csomag, ami relációs adatbázis kezelő rendszert biztosít. Relációs adatbázis rendszereket arra használjuk, hogy a felhasználó által meghatározott rekordokat nagy méretű táblákban tároljunk. Az adat tárolás és kezelés

mellett, az adatbázis motor komplex lekérdezéseket dolgoz fel. Néhány meghatározó funkciója az SQLite adatbázisnak:

1. **Szerver mentes.** Az SQLite-nak nincs szüksége külön szerver folyamatra vagy rendszerre a működéshez. Az SQLite könyvtár közvetlenül kezeli a tárhely fájlokat.
2. **Nulla konfiguráció.** Ha nincs szerver, nincs konfiguráció sem. Úgy csinálhatunk SQLite adatbázist, mintha fájlt hoznánk létre.
3. **Platform független.** Az egész adatbázis példány egy darab platform független fájl, ami semmilyen adminisztrációt nem igényel.
4. **Önálló.** Egy darab könyvtár tartalmazza az egész adatbázis rendszert, ami közvetlenül az alkalmazásba integrálódik.
5. **Tranzakciós.** Lehetővé teszi a biztonságos hozzáférést más szálakból vagy folyamatokból.
6. **Kifejezetten megbízható.** A SQLite fejlesztői csapata nagyon komolyan veszi a forráskód tesztelését.

Összességében az SQLite egy funkcionális és flexibilis relációs adatbázis környezet, ami csak minimális erőforrást igényel. [7] Nem mellesleg ez a legtöbbet használt adatbázis motor a világon. Számtalan alkalmazás használja és Android platformon is ez az alapértelmezett adatbázis. Ha többet szeretnénk megtudni róla, akkor azt megtehetjük a <https://www.sqlite.org/index.html> oldalon.

3.3.3. Pusher Beams (Python Server SDK)

A szerverről való értesítés küldéshez a Pusher Beams SDK¹-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. Erről bővebben olvashatunk a <https://docs.pusher.com/beams> oldalon. Ha kimondottan a szerver oldali python dokumentációt szeretnénk elolvasni, azt megtehetjük itt: <https://docs.pusher.com/beams/reference/server-sdk-python>. A szerver és kliens is oldal implementációja is nagyon egyszerű és jól dokumentált, de ha mégis problémánk támadna az online ügyfélszolgálatuk is végtelenül segítőkész.

Bevallom, amikor implementáltam a szerverről való üzenetküldést nem ment minden simán. Az volt a probléma, hogy csak azt az értesítést kaptam meg amelyik nem tartalmazott plussz adatot az értesítés címén és leírásán kívül. Fel kellett keresnem

¹ Software Development Kit

őket, mert hiába követtem a dokumentációkat nem értettem a hiba okát. Az üzenetekre kifejezetten gyorsan (egy napon belül) válaszoltak és kérték, hogy küldjek több információt. Hamar kiderült, hogy a problémát az FCM² okozza, ami nem engedi, hogy egyszerre küldjünk értesítést és adatot. Itt kiegészítésként fontos megemlíteni, hogy a Pusher a Firebase Cloud Messaging által küldi az értesítéseket. Miután rávilágítottak a hiba okára, egyből megoldást ajánlottak és frissítették a dokumentációjukat. Úgy sikerült kikerülni ezt a limitációt, hogy kettő darab értesítést kell küldeni. Az első csak magát az értesítést tartalmazza, a második pedig kizárólag az adatok küldéséért lesz felelős. Természetesen a második értesítésnek nem lesz vizális reprezentációja a kliens oldalon.

3.3.4. Docker

A Docker az egy nyílt forráskódú motor, ami automatizálja az alkalmazások üzembe helyezését konténerekben belül. Úgy lett tervezve, hogy egy helyi súlyú és gyors környezetet nyújtson, amiben hatékonyan tudjuk futtatni a kódunk. Célja még, hogy hordozhatóvá és könnyen buildelhetővé tegye az alkalmazásunk. Szolgáltatás orientált architektúrára ösztönöz és azt ajánlja, hogy minden konténerben csak egy alkalmazás fusson.[8]

Képfájlok

A konténereinket képfájlokból futtatjuk. Rétegezett formátumuk van, ami lépésről-lépésre épülnek fel egy sorozat parancs alapján. Tulajdonságaikat tekintve hordozhatóak, megoszthatóak, tárolhatóak és frissíthetők. A következő paranccsal tudunk képfájlt készíteni (feltéve, hogy a Dockerfile az aktuális könyvtárban van):

```
$ docker build .
```

Minden Docker képfájl rendelkezik egyedi azonosítóval. A példa kedvéért az én képfálm az „edd116d82f90”-es azonosítót kapta, de használatkor nem fontos az egészet kiírni. Megtehetjük azt is, hogy a 9898-as porton futó szolgáltatást a helyi 80-as portra publikáljuk és így futtatjuk. Ezt a következő paranccsal tehetjük meg:

```
$ docker run -p 80:9898 edd11
```

Konténerek

A konténerek lényegében a futó képfájlok példányai. Ha elindítunk egy képfájlt és megnyitunk egy új terminál ablakot, akkor az alábbi paranccsal megtekinthetjük a futó konténert.

² Firebase Cloud Messaging


```
$ docker container ls
```

A parancs kiírja többek között a publikált portot, a konténerben futó parancsot és a képfájl azonosítóját is.

Dockerfile

A szakdolgozatom szempontjából fontos említést tenni a Dockerfile-ról is. A Dockerfile egy DSL³ nyelvet használ útasításokkal, ami segítségével képfájlokat tudunk készíteni. Valójában ez a szöveges dokumentum tartalmazza azokat a parancsokat, amik egy képfájl felépítéséhez/elkészítéséhez szükségesek. Segítségével bárki automatikusan készíthet egy Docker képfájlt, amit aztán egyszerűen lehet futtatni.

Docker Hub

A Docker Hub biztosítja azt a platformot, ahol bárki kereshet és megoszthat konténereket. Kézzel nagyon körülményes lett volna a képfájl verzióinak követése és publikálása, ezért igénybe vettem a Docker Hub szolgáltatását. Ez a szolgáltatás nem más, mint az automatikus buildek készítése. A Docker Hub képes külső forráskódból képfájlt készíteni és azt publikálni. Összetudjuk csatolni a Github projektjeinket a Docker Hub-al. Ezzel hozzájárulva ahhoz, hogy minden egyes változtatásnál a Docker Hub újra buildelje a képfájlt és publikálja a frissített verziót. Az alkalmazás megtekinthető a <https://hub.docker.com/r/tothkris/reversedroid> oldalon.

3.3.5. Apktool

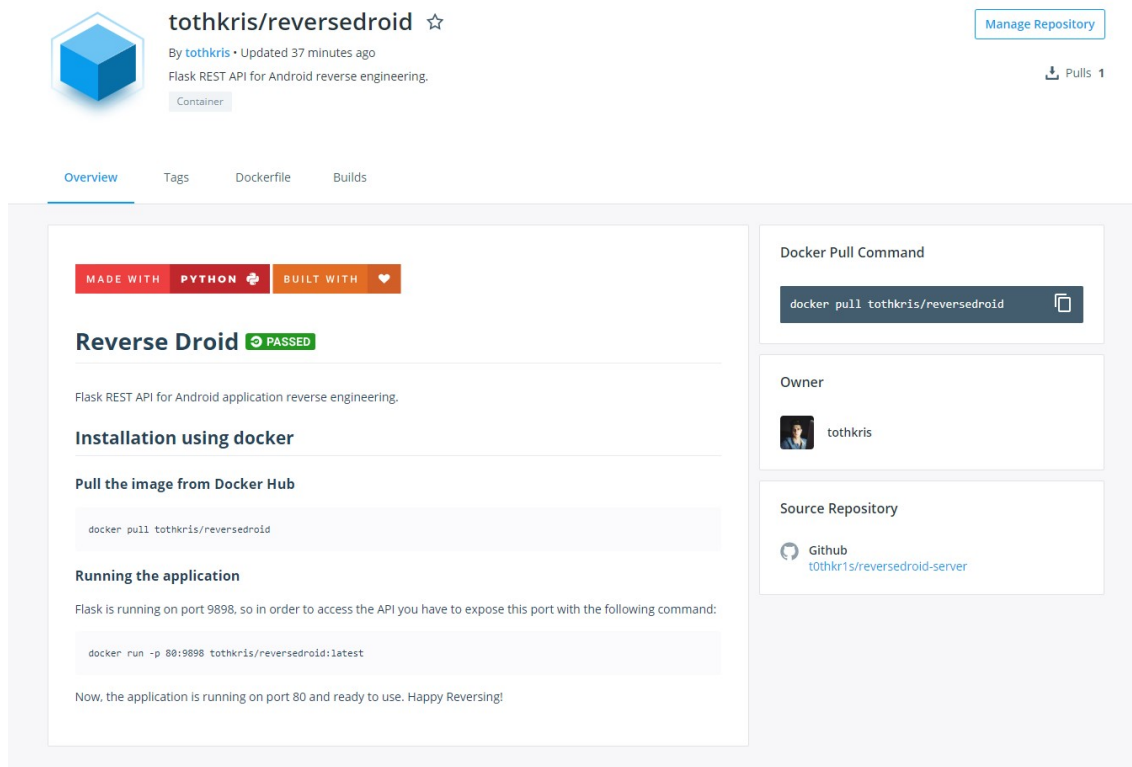
Az Apktool egy olyan eszköz, aminek a segítségével vissza tudjuk fejteni a harmadik féltől származó, zárt forrású, bináris Android alkalmazásokat. Majdnem eredeti az állapotukba tudja dekódolni a forrásokat és akár újra is tudja buildelni az alkalmazást néhány módosítással. Ezek a tulajdonságok vonzák mind a jó és rossz szándékkal rendelkező személyeket. A rossz szándékkal rendelkezőket természetesen azért, mert ezt a technológiát lehet kalózkodásra és egyéb illegális tevékenységre használni. A jó szándékkal rendelkezőket pedig azért, hogy analizálják és megakadályozzák az elterjedését ezeknek a kártékony és illegális szoftvereknek.

Az alábbi tulajdonságokkal rendelkezik:

- Források visszaállítása majdnem eredeti formájukba
- Újra buildelése a dekódolt forrásoknak bináris APK/JAR fájlba
- Smali kód debuggolás

Az eszköz letölthető a <https://github.com/iBotPeaches/Apktool> oldalról.

³Domain Specific Language



3.2. ábra. A publikált applikáció Docker Hub-on.

3.3.6. dex2jar

A szerver oldalon kulcs szerepet játszott a *dex2jar* nevű eszköz. Ahogy a neve is sugallja, az Android *.dex* fájlokat konvertálja *.class* fájlokká. Tehát bináris formátumot alakít egy másik bináris formátumba. 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 ART⁴ számára és ezt kellett átalakítani *.jar* formátumba, amivel már a decompiler dolgozni tud. Szerencsére aktív fejlesztés alatt áll és letölthető a <https://bitbucket.org/pxb1988/dex2jar> oldalról

3.3.7. jd-cmd

A visszefejtéshez kötődik még a *jd-cmd* elnevezésű program. Vannak olyan helyzetek, amikor nem mindig áll rendelkezésünkre GUI⁵. A szerveren például nincs lehetőségünk ilyet használni, ott csak a parancssorra hagyatkozhatunk.

A *jd-cmd* egy Java decompiler, ami *.class* fájlokat alakít vissza *.java* forrásfájlokká. Már ahogy a bevezetésben is leírtam, ezeknek a forrásoknak a visszaállítása nem mindig megfelelő. A visszaállítás sok tényezőtől függ, de ez itt most nem lényeges.

⁴ Android Runtime

⁵ Graphical User Interface (grafikus felhasználói felület)

A <https://github.com/kwart/jd-cmd> oldalon megtekinthetjük és letölthetjük a programot. A használatához szükségünk van a Java 6-os verziójára vagy attól újabbra. Viszonylag sok paramétert adhatunk át, amik segítségével testreszabhatjuk a kimenetet. Ilyenre példa a *-displayLineNumbers*, ami a sorszámokat jeleníti meg a visszafejtett osztályokban.

3.3.8. JSON

A JSON egy platformok közötti adatátvitelre szolgáló formátum. Az adatátviteli formátum egy olyan szöveges formátum, amit különböző platformok közötti adat cserére használnak. Egy másik széles körben használt adatátviteli formátum az XML⁶. Ezek biztosítják a nagyon eltérő rendszerek közötti adatváltást. A JSON egy olyan formátum, amin megegyeztek ezek a rendszerek adat kommunikáció céljából. [10] Kettő struktúrára épül:

1. **Kulcs-érték párok.** Ez a koncepció nagyon elterjedt a számítástechnikában és több programozási nyelvben van megfelelője.
2. **Listába rendezett értékek.** A legtöbb nyelvben megfelel a tömböknek, vektoroknak és listának.

Ezek olyan univerzális adatszerkezetek, amiket gyakorlatilag minden programozási nyelv támogat ilyen vagy olyan formában. Ésszerűnek tűnik tehát, hogy az az adatformátum, amelyet programozási nyelvek kommunikációjához kívánunk használni, szintén ezekre a szerkezetekre épüljön.[11]

```
1 {  
2   "result": "APK uploaded successfully and reverse engineering started!"  
3 }
```

1. . Egy nagyon egyszerű példa a szerverről érkező JSON válaszra.

Emberek számára is könnyen olvasható és írható, ahogy ez a 1 példán is látszik. A szakdolgozatomban is fontos szerepet tölt be, ugyanis a klien-szerver kommunikáció teljes egészében erre alapul. Véleményem szerint sokkal átláthatóbb és egyszerűbb, mint például az XML társa.

⁶ Extensible Markup Language

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 support 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. OkHttp

A modern alkalmazások a HTTP ⁷ protokollal dolgoznak. Segítségével adatokat tudunk küldeni és fogadni. Ha hatékonyan használjuk a HTTP protokolt, akkor gyorsabban tudjuk az adatokat betölteni és sávszélességet spórolhatunk meg.

Az OkHttp egy HTTP kliens, ami alapértelmezetten hatékony. Néhány tulajdonsága:

1. Az átlátszó GZIP csökkenti a letöltés méretét.
2. A válaszokat cache-ben tárolja, ezért ismétlődő kéréseknél elkerülhető a hálózat használata.

Akkor sem omlik össze, ha problémás a hálózat, valamint csendben helyreáll a gyakori kapcsolati problémák esetében is. Ha a szolgáltatásunknak több IP címe lenne, akkor az alternatív IP címekkel próbálkozna kapcsolati hiba esetén. Ez szükséges IPv4+IPv6 használatakor és olyan szolgáltatások esetében, amik redundáns adat központokban vannak. Támogatja a modern TLS funkciókat, ami a biztonságos adatátvitelért felelős.

Az OkHttp használata egyszerű. A kérés/válasz API-ja builder tervezési mintára épült, amit nagyon könnyű használni.

⁷HyperText Transfer Protocol

Lehetőséget ad szinkronizált (egyidejű) blokkoló hívásokra és aszinkron hívásokra is „callback”-ek segítségével. A könyvtár támogatja az Android 5.0+ (API szint 21+) verziókat és támogatja a Java 8-at, valamint az attól felfelé lévő verziókat. Az OkHttp hivatalos dokumentációja megtekinthető a <https://square.github.io/okhttp/> oldalon.

3.4.3. Pusher Beams (Java Client SDK)

A Pusher Beams SDK-járól már némi említést tettem a Pusher Beams (Python Server SDK) alfejezetben, csak ott a szerver oldalon volt a lényeg. Bonyodalommentessé teszi az eszköz tokenek kezelését és az interakciót az Apple és a Google által biztosított üzenetküldő szolgáltatásokkal.

A Pusher nagyon sokoldalú API-val és SDK-kal rendelkezik. Ezek közül a Beam egy fejlesztő barát eszköz, aminek a segítségével értesítéseket küldhetünk. Számos mobil automatizálási eszköz javasolja a promóciós értesítések küldését, amiket nem mindig szeretnek a felhasználók és leiratkoznak.

A Beam-mel egyenesen a programból indíthatjuk az értesítések küldését a valós alkalmazásban történő eseményekre alapozva. Így a felhasználók kevésbé fogják megenni az értesítéseket is törölni esetlegesen az appot. Íme néhány példa a felhasználást illetően:

- Étél kiszállítással kapcsolatos folyamatos értesítések
- Játékokban pontok alapján való értesítés küldés
- Tranzakciós értesítések

A felhasználásra nekem ezektől eltérő tervem volt, de alapvetően hasonló célra alkalmaztam. Én a felhasználókat az elkészült fájlokról értesítettem, ami kulcs szerepet játszik a programban. A szerverről érkező értesítésben adatokat is küldök, amik nélkül nem lehetne letölteni a fájlokat. Ebből is látszik, hogy nagy mértékben erre a szolgáltatásra épül a szoftver.

3.4.4. Room

A Room egy absztrakciós réteget nyújt az SQLite adatbázishoz, ezzel lehetővé téve az egyszerűbb és hatékonyabb adatbázis elérést. 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. A Room használatakor annotációk segítségével tudjuk összekötni a Java POJO⁸ osztályokat az SQLite adatbázist.

⁸ Plain Old Java Object

3 fő komponense van:

1. **Adatbázis.** Az *@Database* annotációval rendelkező osztálynak az alábbi feltételeket kell kielégítenie.
 - a) Absztrakt osztálynak kell lennie, ami a *RoomDatabase* osztályból származik.
 - b) Tartalmaznia kell az adatbázishoz tartozó entitás listát az annotáción belül.
 - c) Rendelkezik egy olyan metódussal, aminek nincsen paramétere és visszatér egy *@Dao*-val annotált osztállyal.
2. **Entitás.** Egy táblát reprezentál az adatbázisban.
3. **DAO.** Az adatbázist elérő metódusokat tartalmazza..

Az alkalmazás a Room adatbázist használja, hogy megkapja az adatbázishoz társított DAO-kat. Ezután az alkalmazás, ezeket a DAO-kat használja az entitások adatbázisból való eléréséhez. A megkapott entitásokat ért változtatásokat egyszerűen vissza tudjuk menteni az adatbázisba. Végül, az alkalmazás entitást használ, hogy lekérjen és beállítson értékeket a tábla megfelelő oszlopaiba az adatbázison belül.[12]

A 2 kódrészletben kiemeltem egy DAO⁹ interfészt. Lényegében, ez a DAO nyújtja a hozzáférést az adatbázis réteghez. Jól látható, hogy annotációkkal kell megjelölni a metódusokat, annak függvényében, hogy milyen utasítást szeretnénk végrehajtani. Ahol paraméterben SQL¹⁰ utasítást látunk, az mind fordítási időben ellenőrizve van. Ez azért nagy segítség, mert nem kell aggódnunk szintaktikai hibákon, illetve azon, hogy az adott tábla nem létezik. A 2 kódrészletben látható egy *listNotifications(String username)* metódus, aminek a paramétere szerepel az SQL utasításban is egy kettőspont után. A Room elvégzi ezeknek a paramétereknek az összehasonlítását és hibát dob, ha eltérést talál. Lehetőségünk van több paramétert is átadni, ezáltal robosztusabb lekérdezéseket valósíthatunk meg.

3.4.5. CodeView

A forráskód megjelenítését nem lett volna célszerű nulláról felépíteni, ezért inkább kész megoldások után néztem. Kutatásaim során, nem találtam megfelelő natív Android komponenst, amivel kivitelezhető lett volna a forráskód megjelenítése. Ezt úgy értem, hogy a kijelölést nem lehet megoldani egy egyszerű *TextView* osztállyal. Mindenképpen kellett egy olyan könyvtár, ami megbízható és elegendő funkcionalitást és testreszabhatóságot nyújt. Jó pár könyvtárat végig kellett próbálnom, mire ráakadtam az igazira. Találkoztam olyannal, amelyik nem megfelelően töltötte be a kódot és nem is volt

⁹ Data Access Object

¹⁰ Structured Query Language

```

1  @Dao
2  public interface NotificationDao {
3      @Insert
4      void insertNotification(Notification notification);
5
6      @Query("select * from Notification where username=:username")
7      List<Notification> listNotifications(String username);
8
9      @Delete
10     void deleteNotification(Notification notification);
11
12     @Query("delete from Notification")
13     void deleteNotifications();
14 }

```

2. . A NotificationDao interfész.

megbízható. Voltak olyanok is, amelyeket nagyon régóta nem voltak karban tartva és elavult kód bázissal rendelkeztek. Találtam aktív fejlesztés alatt lévőket is, de azok többnyire még kiforratlan állapotban voltak és nem rendelkeztek elegendő funkcióval.

A megvalósítást ezek a könyvtárak, úgy végezték el, hogy lényegében egy *WebView* komponensbe töltötték be a forráskódot, amit aztán különböző JavaScript keretrendszerek segítségével színezték ki. A választásom egy olyan könyvtárra esett, ami a *highlight.js* keretrendszert használja. Ez biztosít több különböző témát szintaktikai kijelölésre és amit még fontosabbnak tartok, hogy automatikusan felismeri a domináns programozási nyelveket, mint a *Java*, *Python* vagy *Ruby*. Így sokkal kényemesebb a használata, mert nem kellett kiterjesztés alapján megadni az adott nyelvet. Néhány funkció hiányzott ugyan, mint például a két ujjal való nagyítás, kicsinyítés és a kezdő betű méret beállítása. Ezeknek a támogatását sikerült viszonylag egyszerűen megoldani, hiszen a *CodeView* a *WebView* osztályból származik és rendelkezik a megfelelő metódusokkal. A 3 kódrészletben a *CodeView* megvalósítása és kiegészítése látható, alatta pedig a 3.3 képen látható a forráskód megjelenítése az alkalmazásban.

A választott könyvtár elérhető: <https://github.com/Thereisnospon/CodeView>

```

1  codeView = view.findViewById(R.id.codeView);
2  codeView.setTheme(CodeViewTheme.RAILSCASTS).fillColor();
3  codeView.getSettings().setBuiltInZoomControls(true);
4  codeView.getSettings().setDisplayZoomControls(false);
5  codeView.getSettings().setTextZoom(50);
6  codeView.setWebViewClient(new WebViewClient() {
7      @Override
8      public void onPageStarted(WebView view, String url, Bitmap b) {

```

```

9         super.onPageStarted(view, url, b);
10        Log.d(Constants.TAG, "onPageStarted: loading page");
11    }
12
13    @Override
14    public void onPageFinished(WebView view, String url) {
15        super.onPageFinished(view, url);
16        Log.d(Constants.TAG, "onPageFinished: page loaded");
17        loading.setVisibility(View.GONE);
18    }
19    });

```

3. . A CodeView könyvtár felhasználása és kiegészítése.



3.3. ábra. A visszafejtett forráskód megjelenítése szintaktikai kijelöléssel.

3.4.6. Espresso

Az Android Studio rendelkezik jó pár funkcióval, amikről már tettem említést. Amit szándékosan kihagytam, az a UI¹¹ tesztek felvétele. Az Espresso Test Recorder funkció által megtehetjük, hogy nem kézzel írjuk a teszteket, hanem felvesszük őket. Feltudjuk venni az interakcióinkat az alkalmazással és ellenőrizni tudjuk az elemeket a felhasználói felületen, valamint biztosítani tudjuk az adott pillant kép helyességét. Az Espresso Test Recorder veszi az elmentett teszt felvételt és automatikusan generál egy hozzátartozó UI tesztet, amit aztán futtatni tudunk az alkalmazásunk tesztelése érdekében.

Espresso Test Recorder az Espresso Testing keretrendszer alapján írja a teszteket, ami egy API az AndroidX Test-ben. Segít megbízható és tömör UI teszteket írni a felhasználói interakcióra alapozva. Állíthatunk elvárásokat és interakciókat anélkül, hogy hozzáférnénk az alkalmazás nézeteihez és activity-jeihez. Ez a struktúra optimalizálja a tesztek futási idejét. [4]

A projektem esetében nem volt ilyen egyszerű a helyzet. Mindenképpen segített, hogy feltudtam venni a teszteket, de vegyük a következő helyzetet. Tegyük fel, hogy a bejelentkezés folyamatára már készen van a teszt. Amikor Espresso-val tesztet veszünk fel, az alkalmazás tiszta állapotról indul. Ez az jelenti, hogy ha bármilyen bejelentkezés utáni interakciót szeretnénk felvenni, akkor a teszt a bejelentkezéssel együtt kezdődik. Az ilyen problémákat manuálisan kell kikerülni. A 4 kódrészletben megfigyelhető, hogy egy *@Before* annotációval ellátott metódusban elmentettem egy teszt felhasználó nevet és jelszót. Ezután pedig egyszerűen egy Intent segítségével elindítottam az Activity-t. Mindez nem volt elegendő ahhoz, hogy sikeresen tovább mehessek ugyanis bejelentkezés után történik a különböző engedélyek ellenőrzése. Az Espresso arra is nyújt megoldást, hogy ezeket az engedély kéréseket automatikusan elfogadjuk a tesztekben a *GrantPermissionRule* osztály segítségével.

```
1 @LargeTest
2 @RunWith(AndroidJUnit4.class)
3 public class DeleteDataTest {
4
5     @Rule
6     public ActivityTestRule<MainActivity> mActivityTestRule = new
        ↪ ActivityTestRule<>(MainActivity.class);
7
8     @Rule
9     public GrantPermissionRule mGrantPermissionRule =
10         GrantPermissionRule.grant(
11             "android.permission.READ_EXTERNAL_STORAGE",
```

¹¹ User Interface

```

12         "android.permission.WRITE_EXTERNAL_STORAGE");
13
14     @Before
15     public void setSharedPref() {
16         SharedPrefUtil.saveCredentials(
17             getInstrumentation().getTargetContext(),
18             "test", "test");
19         mActivityTestRule.launchActivity(new Intent());
20     }
21
22     @Test
23     public void deleteDataTest() {
24
25         try {
26             Thread.sleep(5000);
27         } catch (InterruptedException e) {
28             e.printStackTrace();
29         }
30
31         onView(allOf(withId(R.id.delete_data),
32             childAtPosition(
33                 childAtPosition(
34                     withId(R.id.action_bar),
35                     1),
36                     0),
37             isDisplayed()))
38             .perform(click());
39
40         onView(allOf(withId(android.R.id.button1),
41             childAtPosition(
42                 childAtPosition(
43                     withId(R.id.buttonPanel),
44                     0),
45                     3)))
46             .perform(scrollTo(), click());

```

4. . Espresso UI teszt az adatok törléséhez.

3.4.7. Material Design 2.0

Az alkalmazás elkészítésekor próbáltam figyelembe venni a felhasználói interfésszel kapcsolatban támasztott ki nem mondott elvárásokat. Korábbi Play Store-ban publikált alkalmazásaim alapján mondhatom, hogy a felhasználónak az app megjelenése legalább olyan fontos, mint a funkcionalitása. A Google már korábban, az Android 5.0 (API verzió 21) verzióval hozta be az akkoriban még újnak számító Material Design-t és teljesen megreformálta az Android platform felületét. Sok embernek tetszettek az élénk színek és a vetett árnyakok, amik meghatározó részei voltak a komponenseknek. Persze, ez nem volt tökéletes. A nevéből is ered, hogy az anyagokból kiindulva és azok egymásra pakolásával született meg a dizájn. Kicsit pontosabban fogalmazva a Material Design irányelveket foglal össze, amiket nem fontos követni, de ajánlott. A Material Design 2 nem hozott hatalmas eltéréseket elődjéhez képest, inkább csak letisztultabb lett, ami szimplán jobb vizuális élményt nyújt. Az Android platformot tekintve új komponensek érkeztek, ezek közé tartozik a *BottomAppBar* és a *Chip*.

Miután az Android Studio-ban létrehozunk egy üres Android projektet az nem fogja támogatni a új Material Design 2.0 könyvtárat. Annak érdekében, hogy ezen a kisebb problémán túllépjek hozzá kellett adnom néhány függőséget a projekthez és át kellett refaktorálnom AndroidX kompatibilissá. Ezt nagyon egyszerűen megcsinálhatjuk, ugyanis az Android Studio fejlesztői környezet biztosít egy ilyen menü pontot. A Google csinált a lépésekről egy kiváló dokumentációt, ami elérhető a <https://material.io/develop/android/docs/getting-started/> oldalon. Ráadásul az oldalon találunk rengeteg hasznos információt a komponensekről és azok használatáról.

4. fejezet

A visszafejtés teszteléséhez felhasznált programok

4.1. Sérülékeny Android alkalmazások

A teszteléshez keresnem kellett olyan sérülékeny alkalmazásokat, amelyeken jól lehet demonstrálni a visszafejtést. Minden alkalmazás, amit itt megemlítek nyílt forráskódú 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 jogi következményei egy ilyen visszafejtésnek elég ködösek. Szerettem volna biztosra menni, ezért is választottam nyílt forráskódú, szabadon elérhető alkalmazásokat teszteléshez, mert így érezhetem magam a legnagyobb biztonságban. A következő lista tartalmazza a teszteléshez felhasznált szándékosan sérülékeny alkalmazásokat.

1. **Damn Insecure and Vulnerable Application (Diva).** Talán kicsit régi, mert több mint 3 éve nem volt frissítve. Ettől függetlenül ezzel értem el a legjobb eredményt és ezt találtam a leghasznosabbnak.
Megtalálható itt: <https://github.com/payatu/diva-android>.
2. **Frida.** Ez egy általam fejlesztett alkalmazás, ami kimondottan dinamikus instrumentáció tesztelésére és bemutatására készítettem.
Megtalálható itt: <https://github.com/t0thkr1s/frida-demo>
3. **Purposefully Insecure and Vulnerable Android Application(Pivaa).** A Pivaa egy kicsit modernebb verziója a Diva alkalmazásának. Rendkívül sok sérülékenységet tartalmaz, amiből sokat lehet tanulni.
Megtalálható itt: <https://github.com/HTBridge/pivaa>

A többi alkalmazást a <https://github.com/as0ler/Android-Examples> repository-ból használtam fel, ami nem csak applikációkat tartalmaz, hanem játékokat is. A for-

ráskód itt nem érhető el, viszont visszafejtés után betekintést nyerhetünk játékok felépítésébe is.

4.1.1. Célközönség

Ezek a sérülékeny alkalmazások több okból jönnek létre és általában a rossz fejlesztői gyakorlatokat mutatják be. Ilyen programok segítségével tesztelhetjük az elméletben elsajátított tudásunkat. Gyakran biztonsági tréningek vagy éppen CTF¹ játékok ürügyén készítenek ilyen szoftvereket. Kiváló tanulási eszköz lehet az Android alkalmazás fejlesztők, behatolás tesztelők és más érdekeltségi körrel rendelkező személy számára.

4.1.2. Sérülékenységek

Kiszeretném emelni és egy picit jobban bemutatni a lista első elemét, vagyis a Diva applikációt. Ezáltal szó fog esni a különböző sérülékenységekről is, amik Android alkalmazásokat érintenek. Ezek mind statikus vizsgálattal felfedhetők, nem kell dinamikus analízist alkalmaznunk. A következő listába szedtem össze néhány fontosabb sérülékenységet.

1. **Nem biztonságos logolás.** Ez mondjuk pont egy olyan hiba, amit megfigyelhetünk az Android logoló rendszerében, ami a *logcat* elnevezést kapta.
2. **Beégetett titkosítási/API kulcsok.** Ezek több helyen is előfordulnak, angolul „hardcoded strings”-nek hívják. Elég súlyos hibának tekinthető, mivel egyszerűen kiolvasható a visszafejtett forráskódból.
3. **Input validálás.** Számos helyen találkoztam SQL injektálással. Ez a típusú hiba webes alkalmazásoknál is gyakran előfordul és a nem megfelelő input validálásra alapul.

¹ Capture The Flag

5. fejezet

Megvalósított funkciók

5.1. Regisztráció

A szolgáltatást nem akartam mindenki számára elérhetővé tenni, csak a regisztrált felhasználóknak. A 5.1 ábrán látható a regisztrációs felület. Igazából semmi különösebb mezőt nem tartalmaz, csak a szokásos adatokat kell megadnia annak, aki igénybe szeretné venni a szolgáltatást. Ha a regisztráció sikeresen befejeződött, akkor vissza irányítom a felhasználót a bejelentkezési felületre. A folyamatot próbáltam minél egyszerűbbre és magától értetődőre tervezni.

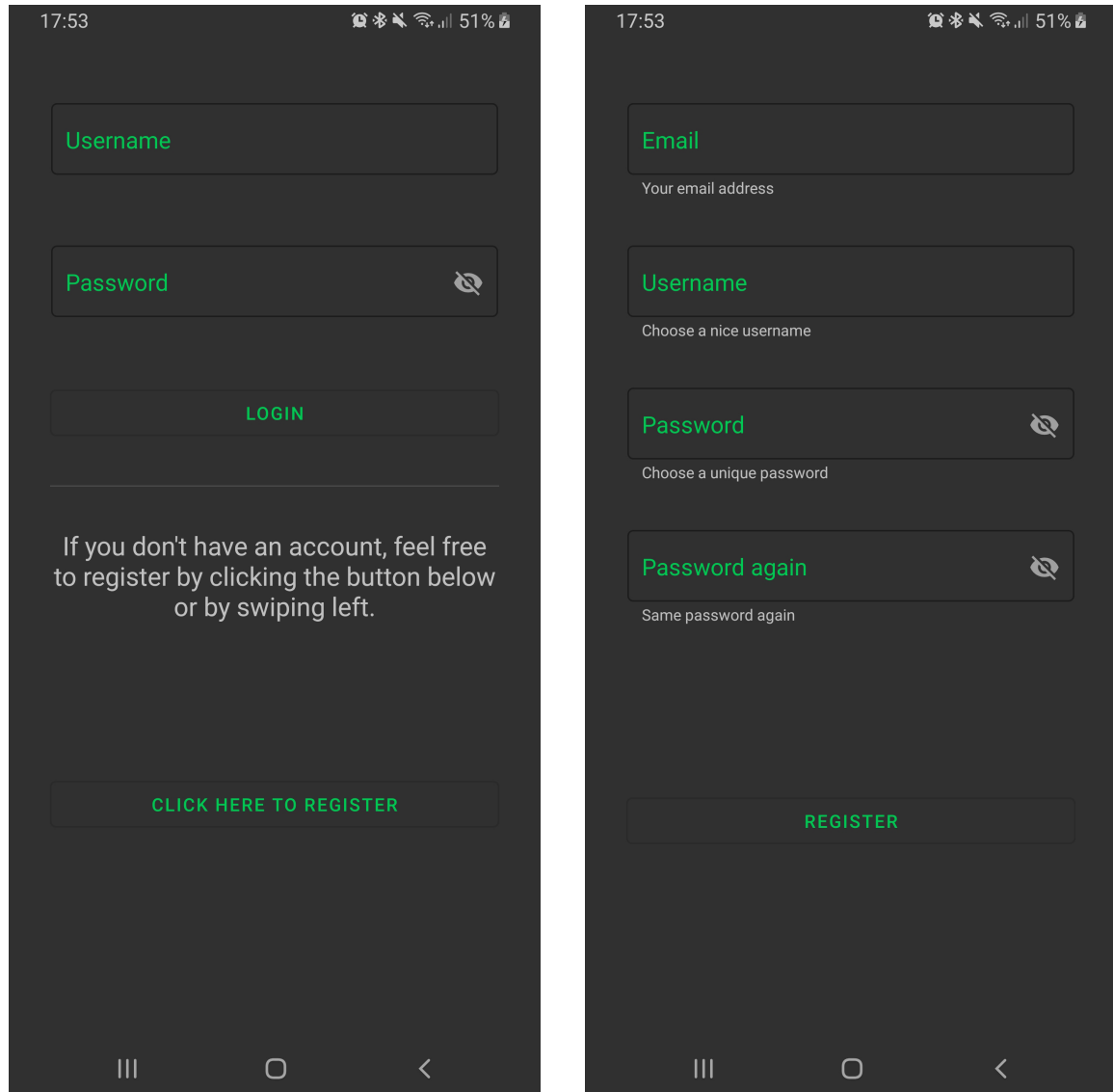
A mezőket tekintve *TextInputLayout*-ba ágyazott *TextInputEditText*-et használtam. A *TextInputLayout* rendelkezik, olyan tulajdonságokkal, mint például a segítséget nyújtó szöveg és az animáció. Ez által még nyilvánvalóbb, hogy melyik mezőbe mit kell írni. Számításba vettem, hogy a felhasználó nem biztos, hogy álló helyzetben fogja tartani a telefonját, ezért külön layout fájlokat készítettem a fekvő nézethez. Szerencsére nem kellett minden felülethez külön fekvő nézetet készítenem, csak a regisztrációhoz és a bejelentkezéshez.

5.2. Bejelentkezés

Természetesen regisztrálás után a felhasználónak be kell jelentkeznie, hogy hozzáférhesen azokhoz a végpontokhoz, amik bejelentkezéshez vannak kötve. Lényegében az összes lényeges végpont bejelentkezéshez van kötve, ezért mondhatjuk, hogy a bejelentkezést kötelező megtenni.

A felület itt csak két mezőt tartalmaz, a felhasználó nevet és a jelszót. Mivel ez egy folyamatban lévő fejlesztés nem tartottam prioritásnak a jelszó visszaállítás funkció implementálását, ebben a korai verzióban. A jelszó mezőben bekapcsoltam egy úgynevezett jelszó kapcsolót, ami a mező jobb szélén látható. Segítségével a felhasználók láthatóvá tehetik a beírt jelszójukat és ellenőrizhetik annak helyességét.

Alapvetően a *MainActivity* indul az alkalmazás futtatásakor és az *AuthenticationActivity* csak akkor indul, ha nincsen mentett felhasználónév és jelszó páros. Az *AuthenticationActivity* tartalmazza mind a bejelentkezési, mind a regisztrációs fragment komponenst.



5.1. ábra. A bejelentkezési és regisztrációs felület megvalósítása.

5.3. Kijelentkezés

Szerver oldalon csak az *Authorization Header* van ellenőrizve, ami azt jelenti, hogy minden egyes hálózati kérésben benne kell hogy legyen. Ennek a kezelését úgy oldottam meg, hogy sikeres bejelentkezés után *SharedPreferences* használatával eltároltam a felhasználónevet és a jelszót. Amíg nem történik kijelentkezés, addig ezek az adatok mentve maradnak.

A 5 kódrészleten láthatjuk, hogyan történik az adatok mentése. A *SharedPreferences* segítségével kulcs-érték párokat tárolhatunk biztonságos módon. A *getSharedPreferences()* metódus 2 paramétert vár, az egyik a név, a másik a használni kívánt mód. Névnél az alkalmazás csomag nevét választottam és privát módban hoztam létre, hogy csak az én alkalmazásom férhessen hozzá az adatokhoz.

```
1 public static void saveCredentials(@NonNull Context context, String
   ↪ username, String password) {
2     SharedPreferences.Editor editor =
       ↪ context.getSharedPreferences(context.getPackageName(),
       ↪ Context.MODE_PRIVATE).edit();
3     editor.putString(Constants.USERNAME_KEY, username);
4     editor.putString(Constants.PASSWORD_KEY, password);
5     editor.apply();
6 }
```

5. . Adatok mentése *SharedPreferences* segítségével.

A 6 kódrészletben látható metódus nem sok eltérést mutat, viszont elengedhetetlen volt ahhoz, hogy töröljem az összes *SharedPreferences*-be mentett adatot.

```
1 public static void deleteCredentials(@NonNull Context context) {
2     SharedPreferences.Editor editor =
       ↪ context.getSharedPreferences(context.getPackageName(),
       ↪ Context.MODE_PRIVATE).edit();
3     editor.clear();
4     editor.apply();
5 }
```

6. . Az összes *SharedPreferences*-be mentett adat törlése.

5.4. Adatok törlése

A kliens oldalon elég sok adat jelenik meg, ami természetesen tárolva van. Ilyen például a feltöltések listája, értesítések és a letöltött fájlok is ide tartoznak. Az adatok törlése funkcióval biztosítottam, hogy a felhasználónak lehetősége legyen törölni minden adatot és előzményt. Ide tartozik a gyorsítótárba másolt feltöltött alkalmazások, amik egy idő után értékes tárhelyet foglalhatnak el. Nem beszélve arról, hogy az adatbázisban lévő összes tábla adatát ürítem. Lehetséges, hogy kényelmesebb lett volna, ha a felhasználó maga választja ki mit szeretne törölni. Sajnos ez a korai verzió csak az együttes törlést biztosítja. Jól jöhet ez a funkció akkor is ha a telefon esetleg nem rendelkezik elég tárhellyel vagy egyszerűen csak helyet szeretnénk felszabadítani.

5.5. Fájl feltöltés

Az alkalmazás egy alapvető funkciója, hogy fel tudjuk tölteni az APK fájlokat. Ezt a funkciót próbáltam a legkézenfekvőbb helyen elhelyezni, mert ezzel indul az egész folyamat és magától értetődőnek kell lennie. Ezért lett a legeleső fragment a feltöltések listáját tartalmazó fragment. Ebben a feltöltési listában jelenítem meg a feltöltött APK fájlok neveit és a feltöltés dátumát.

A feltöltéshez a felhasználónak egy jól látható *FloatingActionButton* komponensre kell kattintania. A tervezési szakaszban úgy gondoltam, hogy hasznos lenne egy saját fájl kiválasztó felületet készíteni. Aztán hamar rájöttem, hogy teljesen felesleges újra felalálni a spanyol viaszt és nem kellene haszontalan felületekkel bonyolítani az applikációt. Biztos találkoztunk már olyan alkalmazással, ami elindított egy külső alkalmazást csak hogy elvégezze valamilyen feladatot. Én is egy ilyen megoldást alkalmaztam. Nem csak *Activity*-ket tudunk indítani *Intent*-ek segítségével, hanem különböző műveleteket is elvégezhetünk. Az ilyen *Intent*-eket *Implicit Intent*-eknek nevezzük. Példának okáért, a fájl kiválasztás egy ilyen művelet. *Implicit intent* használatával indítottam el a fájl kiválasztást, aminek többek között megadtam, hogy milyen típusú fájlokat szeretnék kiválasztani. Ez a típus természetesen a *application/vnd.android.package-archive* volt, ami megfelel az Android telepítő fájloknak. Ebben az esetben nincs lehetőségünk más típusú fájlt kiválasztani csak a megadottat.

A fájl kiválasztása után az *onActivityResult* metódus *Intent* típusú paraméterében kapjuk meg az URI¹-t. Ezt az URI-t felhasználva elmentettem a fájlt az alkalmazás gyorsítótárába. Erre azért volt szükség, mert különben nem kaptam volna meg az abszolút elérési útvonalat. Eztán már csak service elindítása maradt hátra.

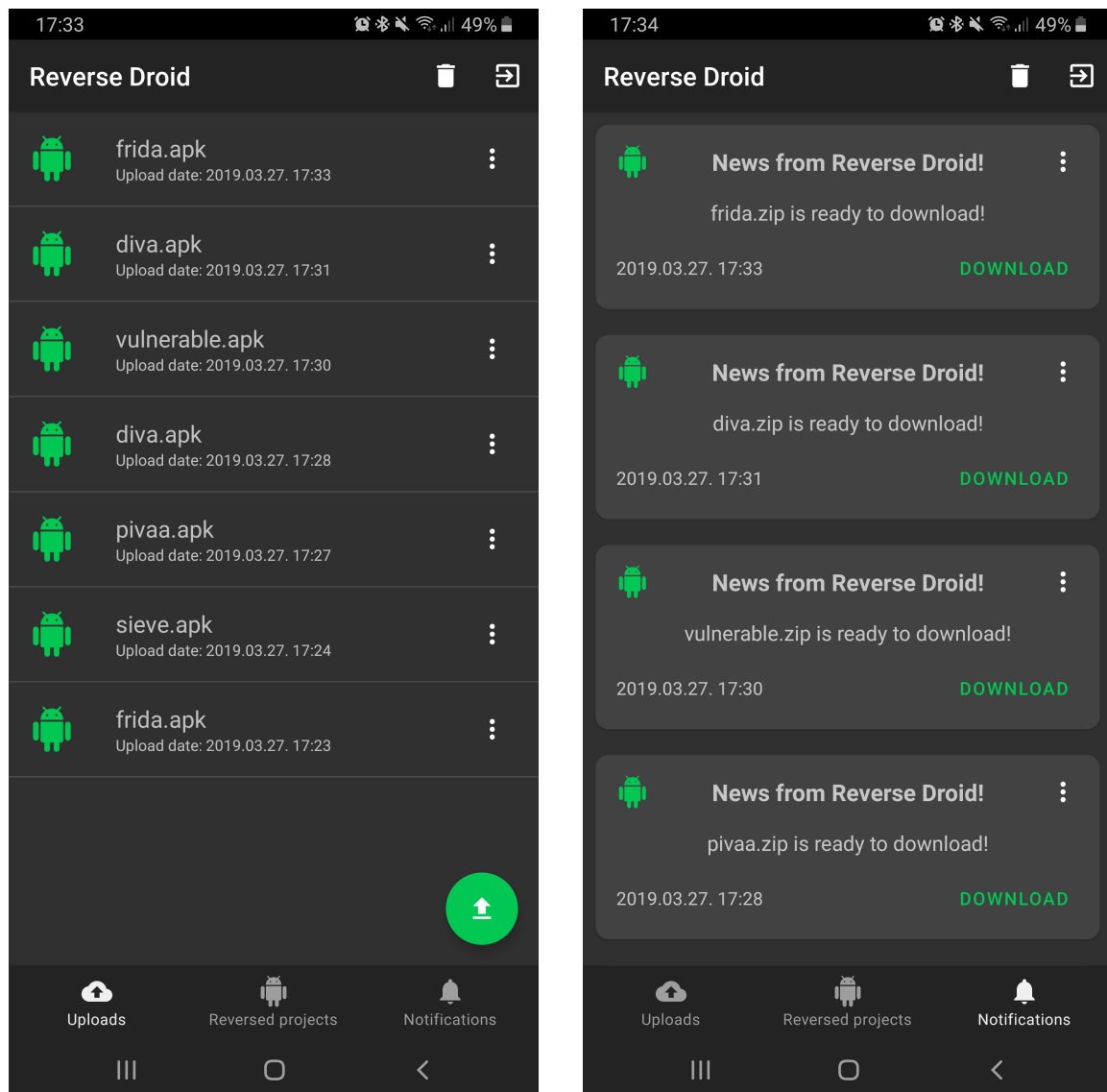
A *startService(Intent service)* metódus egy *Intent*-et vár, ami segítségével elindíthatjuk a háttér service-t. Fontos, hogy ez még így nem elég, hanem extraként hozzá kell adni a feltölteni kívánt fájlt. A 5.2 ábra bal oldali képernyőképén látható a feltöltött fájlok listája.

5.6. Fájl letöltés

A fájl letöltés is egy roppant fontos funkció, ugyanis csak így jut el a felhasználóhoz az elkészült projekt. A megvalósítását hozzákötöttem az értesítésekhez, mert a kliens abban kapja vissza a letöltendő fájl nevét. Megfigyelhetjük a 5.2 ábra jobb oldali képernyőképén, hogy minden egyes értesítés kártyáján ott a letöltés gomb. Ha ezekre rákattintunk, akkor itt is minden egyes alkalommal egy újabb háttér service fog elindulni. Azt is megtehetjük, hogy akár többször is letöltjük ugyanazt a fájlt. Ha már lokálisan létezik egy ugyanolyan nevű projekt, mint amit éppen letöltünk akkor sem kell

¹ Uniform Resource Identifier

félnünk, hogy felül írjuk, mert egyedi nevet fog kapni. Ezt úgy valósítottam meg, hogy ha létezik már a projekt mappája, akkor hozzáfűzök egy számot és 0-tól kezdődően mindig növelem.



5.2. ábra. A feltöltött APK fájlok és értesítések felülete.

5.7. Navigáció a fájlrendszerben

A letöltött és kitömörített projektek az alkalmazás mappájában kerülnek. Ennek a mappának a neve megegyezik az alkalmazás nevével és az elsődleges külső tárhelyen foglal helyet. Ezzel alapvetően nincs is semmi probléma, csak ha megszeretnénk nézni a fájlokat, akkor el kellene hagynunk az alkalmazást. Ez pedig rossz felhasználói élményhez vezetett volna. Ennek megoldására megvalósítottam egy nagyon egyszerű fájlrendszer navigációt, de ezt csak az alkalmazás mappájában tettem lehetővé. Szóval a

felhasználó nem tud a fájlrendszerben feljebb navigálni, csak a „Reverse Droid” gyökér mappáig. Ezzel a funkcióval kitömörítés után egyszerűen és gyorsan elérhetővé válik a projekt a felhasználó számára. A mappák esetében megjelenítettem a módosítás dátumát és hogy hány fájlt/mappát tartalmaz. A fájlok részleteit tekintve a fájl nevét, módosítás dátumát és a fájl méretét is megjelenítettem.

Ezeket felül hozzáadtam egy olyan funkciót, hogy ha hosszan kattintunk egy mappán, akkor lehetőséget kapunk annak törlésére. Ha nem egy mappán kattintunk hosszan, hanem egy fájlra, akkor három opció közül is választhatunk. Az első triviális funkció itt is a fájl törlése volt, a második a fájl megosztása kompatibilis külső alkalmazással. Végül, de nem utolsó sorban a kód megtekintése menü pont kapott helyet. Ennek a működéséről és megvalósításáról már említést tettem a CodeView alfejezetben, de nem említettem meg minden részletet. Kimaradt például az, hogy csak bizonyos fájl-kiterjesztéseket támogatok jelen helyzetben. Mivel a forráskód tartalmazza az igazán érdekes dolgokat, ezért arra koncentráltam, hogy legalább az Android projektekre jellemző szöveges fájl-kiterjesztéseket támogassam. A különböző képek és speciális fájlok megjelenítése egyelőre nincsen támogatva, de ez is egy olyan funkció, amit megszeretnék valósítani a jövőre nézve. A támogatott kiterjesztéseket jelenleg az is befolyásolja, hogy maga a *CodeView* miket támogat.

A visszalépés megvalósítása okozott egy kisebb fejtörést. A vissza gomb megnyomására túlságosan is bonyodalmas lett volna megvalósítani a visszalépést, ezért úgy döntöttem, hogy egy külön elemmel fogom ezt kivitelezni. Ez az elem úgy mond egy speciális lista elem, amit akkor jelenítek meg ha nem az alkalmazás mappájában tartózkodunk. Emlékeim szerint több fájl kezelő is használta már ezt a megoldást és elég hatékonynak tűnt ez a fajta visszalépési technika.

5.8. Értesítések

Az értesítések küldése és fogadása szintén egy kulcs fontosságú része az alkalmazásnak, ezt már többször is hangsúlyoztam. A fájlok letöltése és kitömörítése mérettől, valamint hálózati kapcsolattól függően időt vesz igénybe. Az elején úgy gondoltam, hogy egy egyszerű *Toast* üzenet is elég a feltöltés állapotának visszajelzésére. Aztán hamar rákellett jönnöm, hogy ez nem ad elég információt a felhasználó számára, ráadásul rossz felhasználói élményhez is vezet. Végül úgy döntöttem, hogy feltöltést is hasonlóan valósítom meg, mint a letöltést.

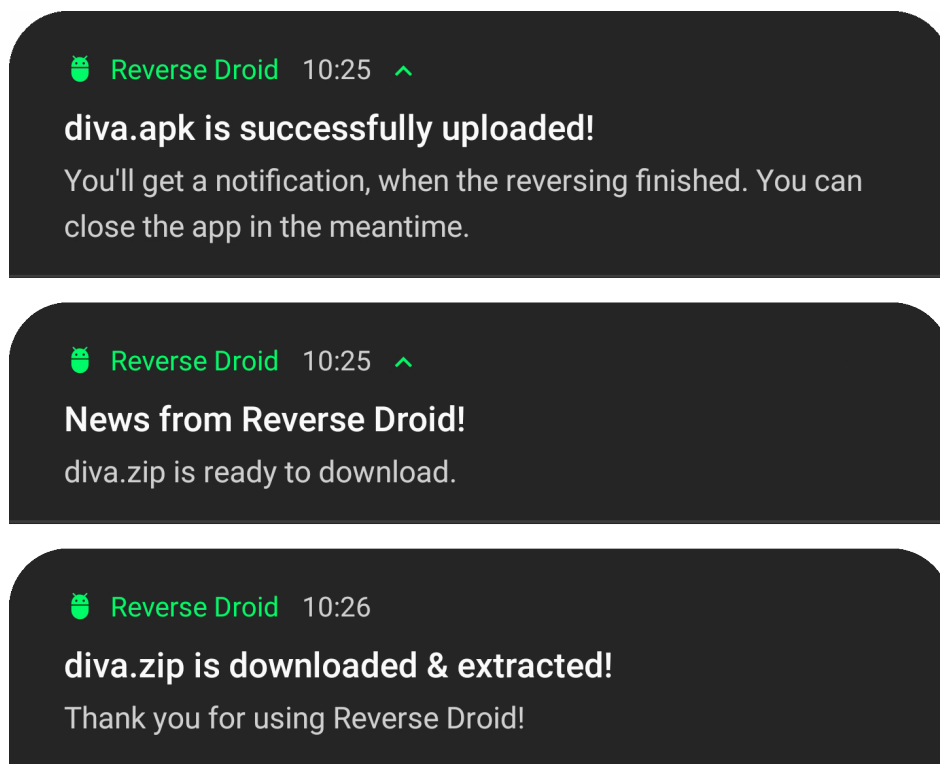
A 5.3 ábrán láthatóak az alkalmazásban használt értesítések. Ez a három értesítés jól érzékelteti a három fő funkcióját az applikációnak. Ezeket össze is foglaltam a következő listában:

1. Android telepítő fájl feltöltése

2. Értesítés a projekt készenlétéről
3. Az elkészült tömörített projekt letöltése

A megvalósítást illetően az első és a harmadik funkció megvalósítása kliens oldalon történt teljes egészében. A harmadik természetesen a szerverről érkezik, ha befejeződött a fájl feldolgozása. Ezek mind az adott körülményektől függnnek, ezért azzal is számolni kell, hogy jelentős időtartamot igényelnek. Azt is lehetővé szerettem volna tenni a felhasználó részére, hogy a műveletek között is elhagyhassa az applikációt.

Itt is a háttér service volt az az applikáció komponens, amit ennek kivitelezésére felhasználtam. Segítségével hosszabb ideig tartó műveleteket tudunk végezni, anélkül hogy a felhasználó interakcióba lépnek az applikációval.



5.3. ábra. Az alkalmazásban megjelenő értesítések.

6. fejezet

Továbbfejlesztési lehetőségek

Úgy gondolom, hogy sokkal nagyobb piaci érték rejlik ebben az alkalmazásban, mint amennyit egyedül sikerült megvalósítanom. A jövőben is szeretném folytatni a fejlesztést és esetlegesen nyílt forráskódúvá tenni a projekteket, hogy mások is közre tudjanak működni a fejlesztésben.

Szeretnék több figyelmet fordítani a biztonságra és hatékonyságra. Gondolok itt a biztonságos kommunikációra TLS-es keresztül és a harmadik féltől származó könyvtárak csökkentésére. Azért lenne érdemes minimalizálni a harmadik féltől származó könyvtárakat, mert nem mindig tudjuk milyen kódot tartalmaznak és mennyire tartják karban a kód bázist.

Jelenleg csak a forráskódok megjelentését támogatja az alkalmazás, de jó lenne ha különböző fájl típusokat is megtudna jeleníteni. Például a képek vagy adatbázisok megtekintése is kritikus lehet egy Android alkalmazás elemzésekor. A kód visszafejtése végén egy összegző report is hasznos lehetne a felhasználó számára, ami tartalmazná a feldolgozott fájlok számát és egy gyors áttekintést nyújtana az adott alkalmazással kapcsolatban.

Egy forráskód elemző integrálása is jelentős előnnyel járhat a többi alkalmazással szemben. Hasonlóan a Google Play Protect-hez, jelezhetnénk a felhasználó számára, ha a feltöltött *apk* malware. Találkoztam több webes *apk* analízálóval, ahol csak az Android komponenseket emelték ki, de az elemzést már a felhasználóra bízták. Nyilván nem lenne célszerű mindent egy algoritmusra bízni, de mindenesetre könnyebbséget látok egy átfogó analízisben, ami esetlegesen mestersége intelligenciára alapul.

7. fejezet

Tapasztalatok

Úgy érzem elértem a céloom ezzel a projekttel és sokat sikerült tanulnom az elkészítése alatt. Új technológiákat ismertem meg és használtam. Nem állítom, hogy hibamentes és minden működik, hiszen ez mégis csak egy szoftver, amit folyamatosan karban kell tartani és fejleszteni. A fejlesztés minden fázisában találtam valami kihívást, ami segítette a fejlődésemet és arra ösztönzött, hogy jobban megismerjem az adott technológiát.

Természetesen nem volt minden magától értetődő és jó pár nehézséggel is találkoztam, amiket már korábban meg is említettem. Számomra, ha valami nem sikerült az mindig ösztönzőleg hatott és motivált a tanulásban. Sikerült az idegen technológiák iránt érzett kíváncsiságom is kielégíteni. A reverse engineering világába is betekintést nyertem, ami különösen érdekelt engem.

A fejlesztésnél csak egyetlen kellemetlen dologgal találkoztam. Szerintem sok fejlesztő egyet ért ha azt mondom, hogy a hiányos vagy éppen félre vezető dokumentáció a legnagyobb ellensége. Szerencsére nem sokszor futottam bele, de amikor igen, akkor pár napig álltam a fejlesztéssel. Végül az oldalon egy rendkívül türelmes és segítőkész fiatalember segített a helyes irányba.

Még a problémáival együtt is úgy gondolom, hogy sikerült egy olyan szoftvert készítenem, ami egyedi és megállja a helyét a piacon.

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>
- [3] FOWLER M, FOEMMEL M. Continuous integration. Thought-Works 2006 May 1 http://www.dccia.ua.es/dccia/inf/asignaturas/MADS/2013-14/lecturas/10_Fowler_Continuous_Integration.pdf
- [4] ONLINE: Create UI tests with Espresso Test Recorder <https://developer.android.com/studio/test/espresso-test-recorder>
- [5] SOMASUNDARAM R. Git: Version control for everyone. Packt Publishing Ltd; 2013.
- [6] GRINBERG M. Flask web development: developing web applications with python. " O'Reilly Media, Inc."; 2018 Mar 5.
- [7] KREIBICH J. Using SQLite. " O'Reilly Media, Inc."; 2010 Aug 17.
- [8] TURNBULL J. The Docker Book: Containerization is the new virtualization. James Turnbull; 2014 Jul 14
- [9] ONLINE: PyCharm Features <https://www.jetbrains.com/pycharm/features/>
- [10] BASSETT L. Introduction to JavaScript Object Notation: A to-the-point Guide to JSON. " O'Reilly Media, Inc."; 2015 Aug 5.
- [11] CROCKFORD, DOUGLAS Introducing JSON. json.org, 2009 May 28.
- [12] ONLINE Save data in a local database using Room <https://developer.android.com/training/data-storage/room/index.html>