EGER 1774
ESZTERHÁZY KÁROLY EGYETEM
Matematikai és Informatikai Intézet

Kliens-szerver kommunikáció Android platformon

Készítette
Balajti-Tóth Kristóf
Programtervező Informatikus BSc

Témavezető
Tajti Tibor
Egyetemi adjunktus

Eger, 2019

Tartalomjegyzék

1. Fejlesztői környezetek **5**
1.1. Android Studio 5
1.2. Pycharm Professional Edition 5
1.3. Postman 6

2. Platformok **7**
2.1. A szerver kiválasztása és felépítése 7
2.2. Mobil platform választása 8

3. Felhasznált technológiák **10**
3.1. Verzió kezelés 10
3.2. Folyamatos integrálás 11
3.3. Szerveren használt technológiák 12
3.3.1. Flask 12
3.3.2. SQLite 12
3.3.3. Pusher Beams 12
3.3.4. Docker 12
3.3.5. Tmux 12
3.3.6. jd-cmd 13
3.3.7. Apktool 13
3.3.8. dex2jar 13
3.3.9. JSON 14
3.4. Androidon használt technológiák 14
3.4.1. AndroidX 14
3.4.2. OkHttp 14
3.4.3. Okio 15
3.4.4. Pusher Beams 15
3.4.5. Firebase Messaging 15
3.4.6. Room 15
3.4.7. CodeView 16
3.4.8. Espresso 17
3.4.9. Material Design 2.0 19

4. Teszteléshez felhasznált eszközök és források **20**
4.1. Sérülékeny Android alkalmazások 20

5. Megvalósított funkciók **21**
5.1. Regisztráció 21
5.2. Bejelentkezés 21
5.3. Kijelentkezés 21
5.4. Adatok törlése 21
5.5. Fájl feltöltés 21
5.6. Fájl letöltés 22
5.7. Navigáció a fájlrendszerben 22
5.8. Értesítések 23

6. Továbbfejlesztési lehetőségek **24**

7. Tapasztalatok **25**

Bevezetés

Az szoftver fejlesztés egy nagyon komplex folyamat és rengeteg részletre oda kell figyleni. 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 app -al.

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 és ha igen milyen súlyos a hiba. A leghatékonyabb módszer ha visszafejtjük 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 szúrni a gyakori hibákat.

A szakdolgozatomban Android platformra készült alkalmazások forrás fájlokká való visszaállításáról írok, valamint bemutatom hogyan valósítható meg a kliens-szerver kommunikáció egy REST API és egy Android platforma készült kliens segítségével. A felhasználó egy egyszerű autentikáció után képes lesz *.apk* fájlok feltöltésére, letöltésére és az elkészült projekteben való navigálásra. Hosszabb ideig tartó folyamatok állapotáról és elkészültéről értesítést kap és lehetősége lesz a forrás kód alkalmazáson belüli megtekintésére és megosztására. A projektet „Reverse Droid”-nak neveztem el.

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

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 kacsolat 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 kelleténél jobban meríti az akkumulátorunk. Említésre méltó még a flexibilis build rendszere is, a Gradle. Használatával megtehetjük, hogy külön build típusokat hozzunk 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

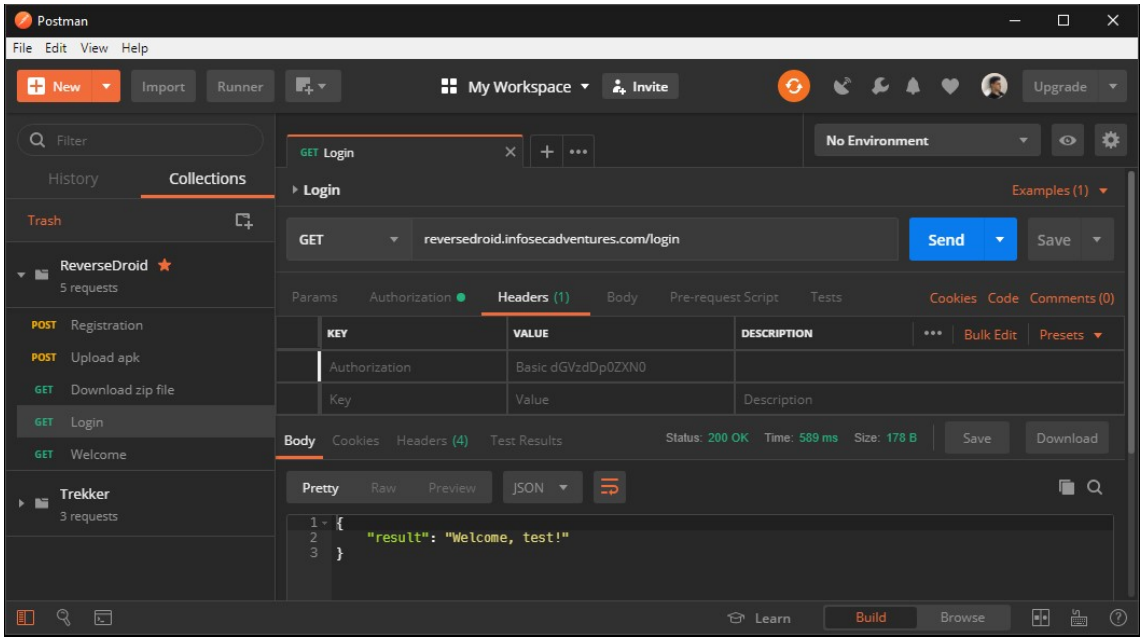
A PyCharm is egy IDE [1], mint az Android Studio. Dolgozhatunk webes technológiákkal vagy mesterséges intelligenciával, a PyCharm megfelelő választás lehet bármilyen területen programozó számára. A Pycharm mögött is a *Jetbrains* cég áll. Ezt azért fontos megemlíteni, mert már 15 éve azon dolgoznak, 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 dolgoznok, 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 rendkívűl praktikus. A fejlesztés közben egyszerűen tudtam feltölteni a szerverre a változtatásaimat. A verzió kezelőknek is egyesített felületet nyújt, amivel jelentős időt spórolhatunk meg. Ez a lehetőség mindkettő fejlesztői környezetben elérhető.[7]

1.3. Postman

Jelenleg a Postman a legnépszerűbb API[2] tesztelésben használt eszköz. A Postman kollekciók 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 csak dokumentáció készítésre és a végpontok tesztelésére használtam. Ettől természetesen sokkal több lehetőség rejlik benne. A 1.1 képen látható egy kérés, ami tartalmaz egy *Authorization Header*-t. Attól függően, hogy helyes-e a felhasználó név és jelszó páros a szerver visszaad egy választ JSON formátumban, amit a kép alján láthatunk.



1.1. ábra. Egy GET kérés és válasz a Postman alkalmazásban.

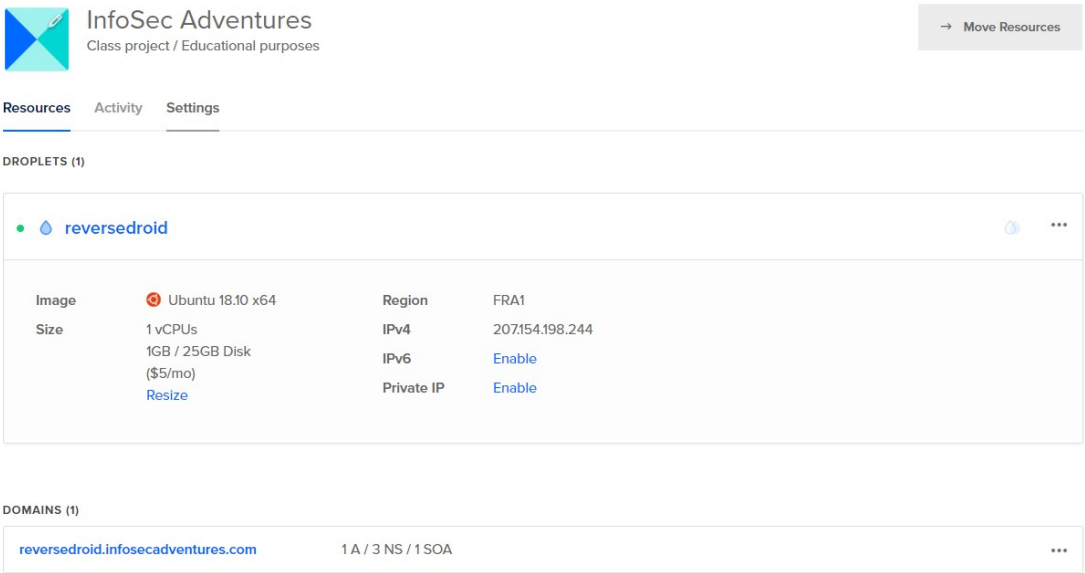
[2] Application Programming Interface

2. fejezet

Platformok

2.1. A szerver kiválasztása és felépítése

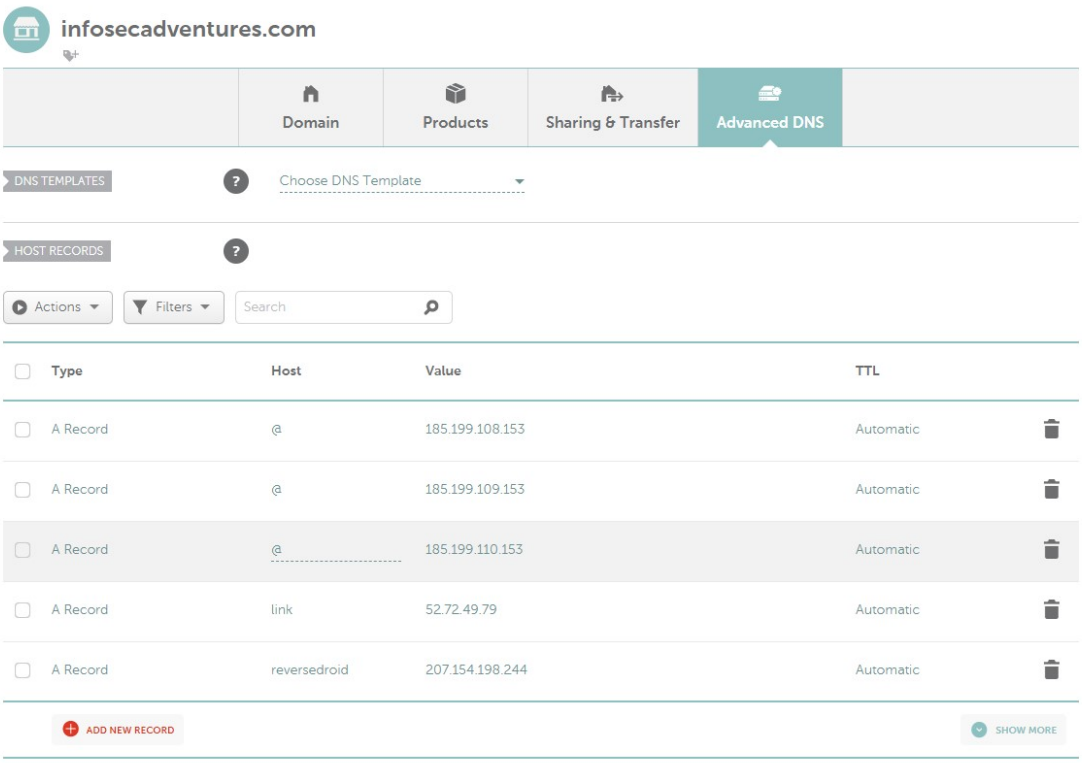
Olyan szerverre volt szükségem, ami nem túl költséges, de mégis megfelelően testreszabható és gyors tárhelyet biztosít. A választásom a Digitial Ocean felhő szolgáltatására esett. Az oldal felületén lehetőségünk van több, úgynevezett *droplet*-et létrehozni, amik nem mások mint virtuális szerverek. Megadhatjuk 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



2.2. ábra. DNS rekordok a domain beállításaiban.

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. 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 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 testreszabhatóság. Az iOS ezzel ellentétben arról híres, hogy egyszerű és megbízható a használata. Viszont jelen vannak olyan megkötések, amik mind a végfelhasználókat és mind a fejlesztőkket limitálják. Gondolok itt a minimális testreszabhatóságra és a nagy részt zárt forráskódra. Itt most nyilván nem szeretném részletezni, hogy melyik

operációs rendszer a jobb vagy éppen rosszab, 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 telefonok terjedt el. Jelen van az okos televíziókban (Android TV), autókban (Android Auto), okos órákban (Wear OS) és IoT[1]-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. Tegyünk 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 testreszabott 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.

[1] Internet of Things

3. fejezet

Felhasznált technológiák

3.1. Verzió kezelés

Egy verzó 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 nagyon nagy 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 biztostja, 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 széleskörben elterjedt a szoftver fejlesztők között. A Git egy ingyenes és nyílt-forrsákódú elosztott verzió kezelő rendszer. Úgy készült, hogy gyorsan és hatákonyan 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.

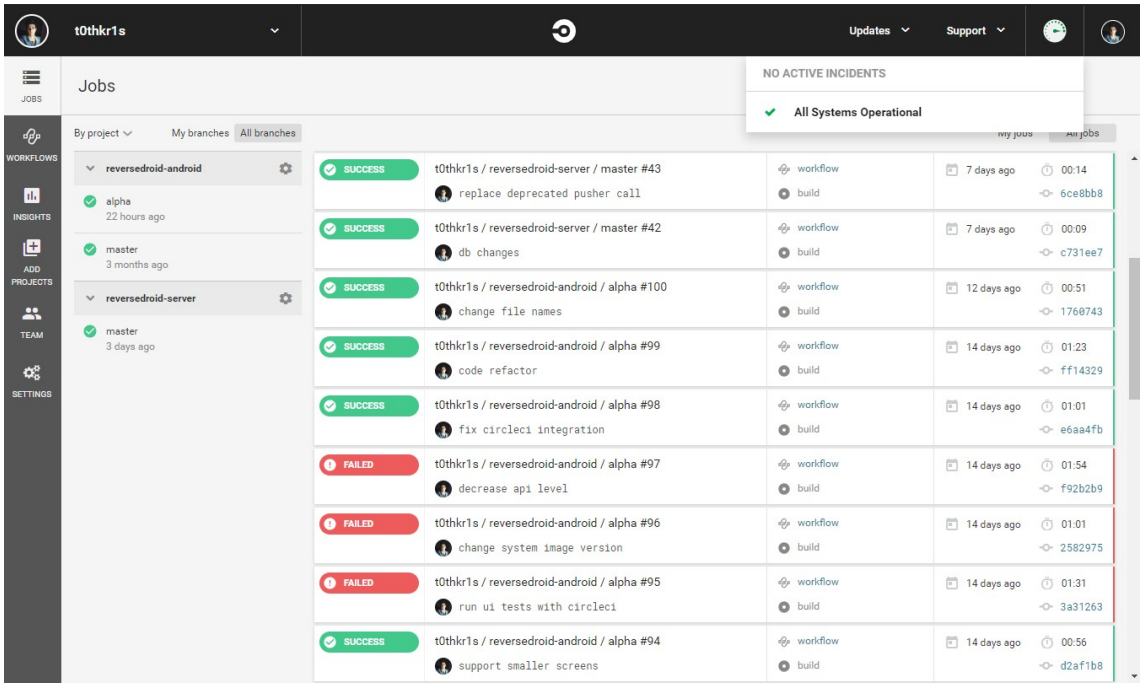
A Github nem összetévesztendő a Git-el, mert a Git a forráskód változtatásainak kezelésére szolgál (lokálisan), a Github pedig egy tárhelyet nyúlt a verzió kezelt könyvtárak és fájlok tárolására. A projekt fejlesztése alatt több gépről dolgoztam és Github

a segítségével biztosítani tudtam, hogy mindenhol elérjen a projektek legfrisseb 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 naponta. 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 a projekteket. 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 futott le sikeresen.



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 programok felépíteni, ami pontosan azt tartalmazza amire szükségünk van és nem tartalmaz semmi fölöslegeset.

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 kiegészítőkön keresztül elérhető. Fejlesztőként lehetőségünk van egyesével kiválogatni vagy éppen magunktól megírni azokat a kiegészítőket, amik az aktuális projekthez kellenek. [6]

3.3.2. 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.

3.3.3. Pusher Beams

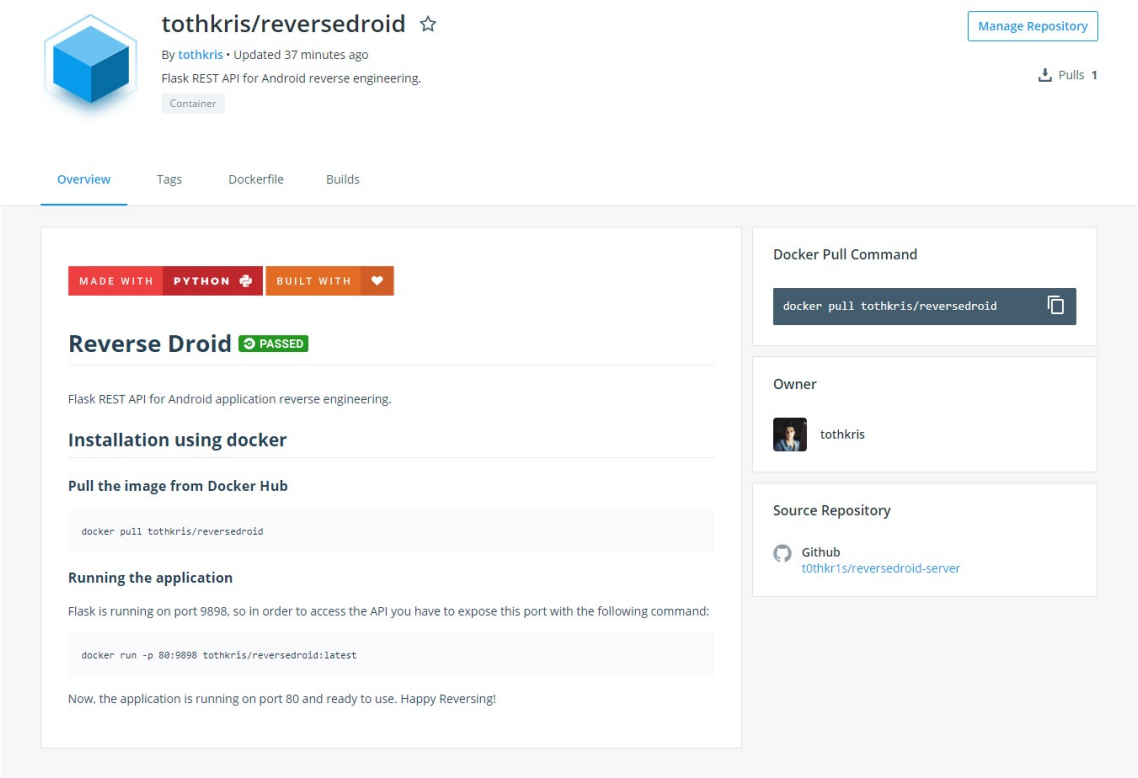
A szerverről való értesítés küldéshez a Pusher Beams SDK-ját használtam. A Beams SDK lehetővé teszi, hogy egyszerűen küldjünk push értesítést „érdeklődési” körök alapján és platformtól függetlenül. Ingyenesen és korlátlanul küldhetjük ezeket az értesítéseket. A szerver és kliens is oldal implementációja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

3.3.4. Docker

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

3.3.5. 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



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

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.

3.3.6. jd-cmd

Szükségem volt egy parancssorból használható Java Decompiler-re is.

`https://github.com/kwart/jd-cmd`

3.3.7. Apktool

`https://github.com/iBotPeaches/Apktool`

3.3.8. 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 *classses.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`

3.3.9. JSON

A JSON[1] egy könnyű, szöveg alapú szabvány vagyis emberek is egyszerűen tudják írni és olvasni. A JavaScript szkriptnyelvből alakult ki egyszerű adatstruktúrák és tömbök reprezentálására. A JavaScripttel való kapcsolata ellenére nyelvfüggetlen, több nyelvhez is van értelmezője. Ezt a szabványt általában szerver és klien kommunikációra/adatátvitelre használják. Véleményem szerint sokkal átláthatóbb és egyszerűbb, mint például az XML[2]. Általánosságban strukturált adatok tárolására, továbbítására szolgál. [8]

3.4. Androidon használt technológiák

3.4.1. AndroidX

Az AndroidX egy nyílt forrsákódú 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 könytárakat. 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 [3] 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.

[1] JavaScript Object Notation
[2] Extensible Markup Language
[3] HyperText Transfer Protocol

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 biztoságot adatátvitelért felelős.

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

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.

3.4.3. Okio

3.4.4. Pusher Beams

A Beams megkönnyíti az éresítések küldését iOS és Android eszközökre. 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 megunni az értesítéseket is törölni esetlegesen az appot. Íme néhány példa a felhasználást illetően:

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

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 egyszerb-

ben 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[4] osztályokat az SQLite adatbázissal.

3.4.7. CodeView

A forrás kód 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 forrás kód megjelenítése. Ezt úgy értem, hogy a kijelölést nem lehet megoldani egy egyszerő *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 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 rendelkezdtek elegendő funkcióval.

A megvalósítást ezek a könytárak, úgy végezték el, hogy lényegében egy *WebView* komponensbe töltötték be a forrás kódot, amit aztán különböző JavaScript keretrendszerek segítségével színeztek 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 3.1 kódrészletben a *CodeView* megvalósítása és kiegészítése látható, alatta pedig a 3.3 képen látható a forrás kód 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 favicon) {
9          super.onPageStarted(view, url, favicon);
10         Log.d(Constants.TAG, "onPageStarted: loading page");
```

[4] Plain Old Java Object

```
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  });
```

3.1. Listing. A CodeView könyvtár felhasználása és kiegésztése.

3.3. ábra. A visszafejtett forrás kód megjelenítése szintaktikai kijelöléssel.

3.4.8. Espresso

Az Android Studio rendelkezik jó pár funkcióval, amikről már tettem említést. Amit szándékosan kihagytam, az a UI[5] tesztek felvétele. Az Espresso Test Recorder funkció által megtehetjük, hogy nem kézzel írjuk a teszteket, hanem felvesszük őket. Feltudjuk venni az interakcióinkat az alkamazással és ellenőrizni tudjuk az elemeket a felhasználói felületen, valamint biztosítani tudjuk az adott pillantkép helyességét. Az Espresso Test Recorder veszi az elmentett teszt felvételt és automatikusan generál egy hozzátartozó UI tesztet, amit aztán futatni 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. Állíhatunk elvárásokat és interakciókat anélkül, hogy hozzáférnénk az alkalmazás nézeteihez és activity-jeihez. Ez a struktúra optimizálja a tesztek futási idejét. [4]

[5] User Interface

A projektem esetében nem volt ilyen egyszerű a helyzet. Mindenképpen segített, hogy feltudtam venni a teszteket, de vegyük a következő helyzetet. Tegyünk 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 3.2 kódrészletben megfigyelhető, hogy egy *@Before* annotációval[6] ellátot 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",
12                     "android.permission.WRITE_EXTERNAL_STORAGE");
13
14     @Before
15     public void setSharedPref() {
16         SharedPrefUtil.saveCredentials(getInstrumentation().
       getTargetContext(), "test", "test");
17         mActivityTestRule.launchActivity(new Intent());
18     }
19
20     @Test
21     public void deleteDataTest() {
22
23         try {
24             Thread.sleep(5000);
25         } catch (InterruptedException e) {
26             e.printStackTrace();
27         }
```

[6] Az annotációk a programkód elemeihez rendelhetők (csomagokhoz, típusokhoz, metódusokhoz, attribútumokhoz, konstruktorokhoz, lokális változókhoz), plusz információt hordoznak a Java fordító ill. speciális eszközök számára.[9]

```
28
29          onView(allOf(withId(R.id.delete_data),
30                  childAtPosition(
31                          childAtPosition(
32                                  withId(R.id.action_bar),
33                                  1),
34                          0),
35                  isDisplayed())).perform(click());
36
37          onView(allOf(withId(android.R.id.button1),
38                  childAtPosition(
39                          childAtPosition(
40                                  withId(R.id.buttonPanel),
41                                  0),
42                          3))).perform(scrollTo(), click());
43      }
44  }
```

3.2. Listing. Espresso UI teszt az adatok törléséhez.

3.4.9. 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 olyab fontos, mint a funkcionalitása. A Google már korábban, az Android 5.0 (API verzó 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 árnyákok, 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*.

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 céltra 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. Regisztráció

A szolgáltatást nem akartam mindenki számára elérhetővé tenni, csak a regisztrált felhasználóknak.

5.2. Bejelentkezés

Természetesen regisztrálás után a felhasználónak be kell jelentkeznie, hogy hozzáférhessen azokhoz a végpontokhoz, amik bejelentkezéshez vannak kötve.

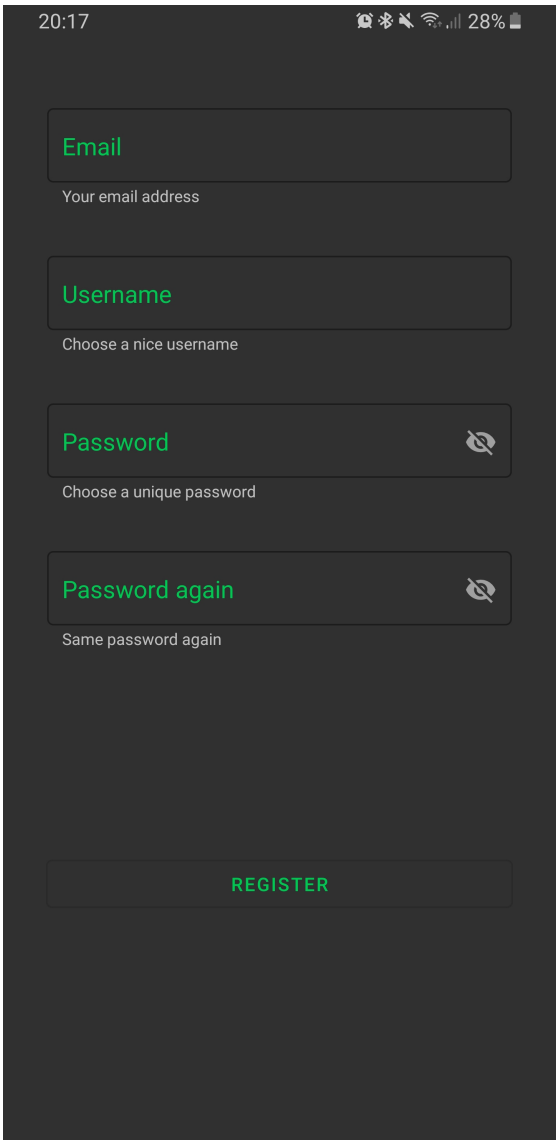
5.3. Kijelentkezés

5.4. Adatok törlése

A kliens oldalon elég sok adat jelenik meg. Ilyen például a feltöltések listája, értesítések és természetesen a letöltött fájlok is ide tartoznak. Az adatok törlése funkcióval biztosítottam, hogy a felhasználónak lehetősége legyen törölni minden adatot és előzményt. 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 meghatározó pontja, hogy az APK fájl feltölthető legyen. Ezt a funkciót próbáltam a legkézenfekvőbb helyre rakni, mert ezzel indul az egész folyamat.



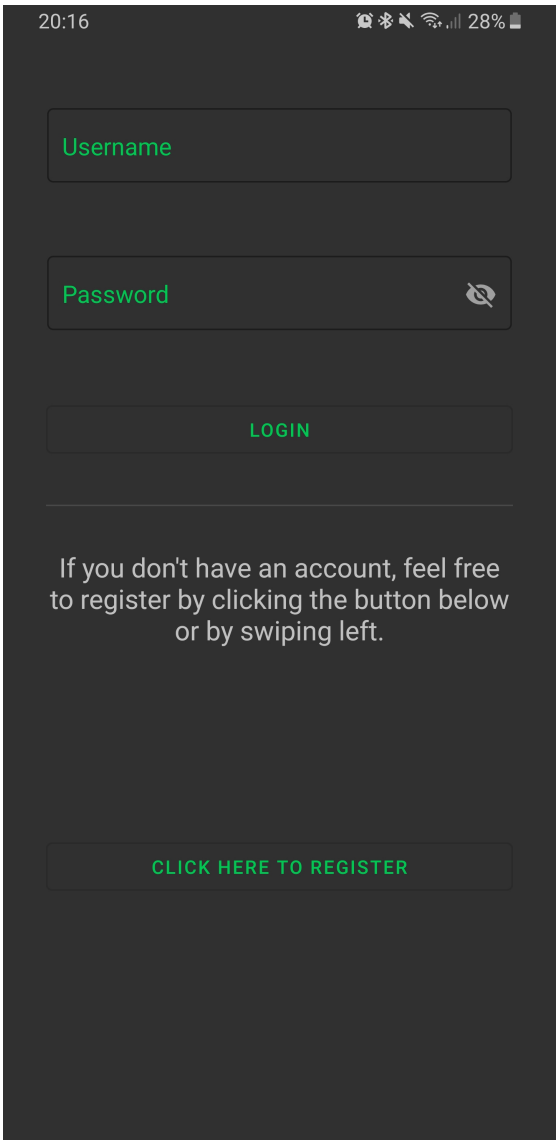
5.1. ábra. A regisztrációs felület megvalósítása.

5.6. Fájl letöltés

A fájl letöltés is egy roppant fontos funkció, ugyanis csak így jut el a felhasználóhoz az eredmény. A megvalósítását hozzákötöttem az értesítésekhez, mert a kliens abban kapja vissza a letöltendő fájl nevét.

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 elkellene 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



5.2. ábra. A bejelentkezési felület megvalósítása.

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, 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 könytárakat, 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 adabázisok megtekintése is kritikus lehet egy Android alkalmazás elemzésekor.s 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 forrás kód 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* analizálóval, 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 célom 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 ösztönzöt, 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 dokumnetáció 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] ONLINE: PyCharm Features https://www.jetbrains.com/pycharm/features/

[8] ONLINE: JSON https://hu.wikipedia.org/wiki/JSON

[9] ONLINE: Java annotációk https://hu.wikipedia.org/wiki/Java_annot%C3%A1ci%C3%B3k