



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	9
3.1. Verzió kezelés	9
3.2. Folyamatos integrálás	9
3.3. Szerveren használt technológiák	10
3.3.1. jd-cmd	11
3.3.2. Apktool	11
3.3.3. dex2jar	11
3.3.4. JSON	12
3.4. Androidon használt technológiák	12
3.4.1. AndroidX	12
3.4.2. OkHttp3	13
3.4.3. Okio	13
3.4.4. Pusher Beams	13
3.4.5. Firebase Messaging	13
3.4.6. Room	13
3.4.7. CodeView	13
3.4.8. Espresso	14
3.4.9. Material Design 2.0	15
4. Teszteléshez felhasznált eszközök és források	16
4.1. Sérülékeny Android alkalmazások	16

5. Megvalósított funkciók	17
5.1. Bejelentkezés	17
5.2. Regisztráció	17
5.3. Kijelentkezés	17
5.4. Adatok törlése	17
5.5. Fájl feltöltés	17
5.6. Fájl letöltés	17
5.7. Navigáció a fájlrendszerben	17
5.8. Értesítések	18
6. Továbbfejlesztési lehetőségek	19
7. Tapasztalatok	20

Bevezetés

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

A statista.com [1] kutatása szerint 2020-ra több mint 4.78 billió telefon lesz használatban. Ezzel a cégek is tisztában vannak és tudják, hogy ha még több emberhez szeretnék eljuttatni a szolgáltatásukat, akkor rendelkezniük kell saját mobilos 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 obfuscált¹ kóddal állunk szemben, de egy tapasztalt szem így is kitudja szűrni 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 állapotról és elkészültéről értesítést kap és lehetősége lesz a forrás kód alkalmazáson belüli megtekintésére és megosztására. A projektet „Reverse Droid”-nak neveztem el.

¹ Az obfuscáció 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 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. Ez a lehetőség mindkettő fejlesztői környezetben elérhető.[4]

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 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.

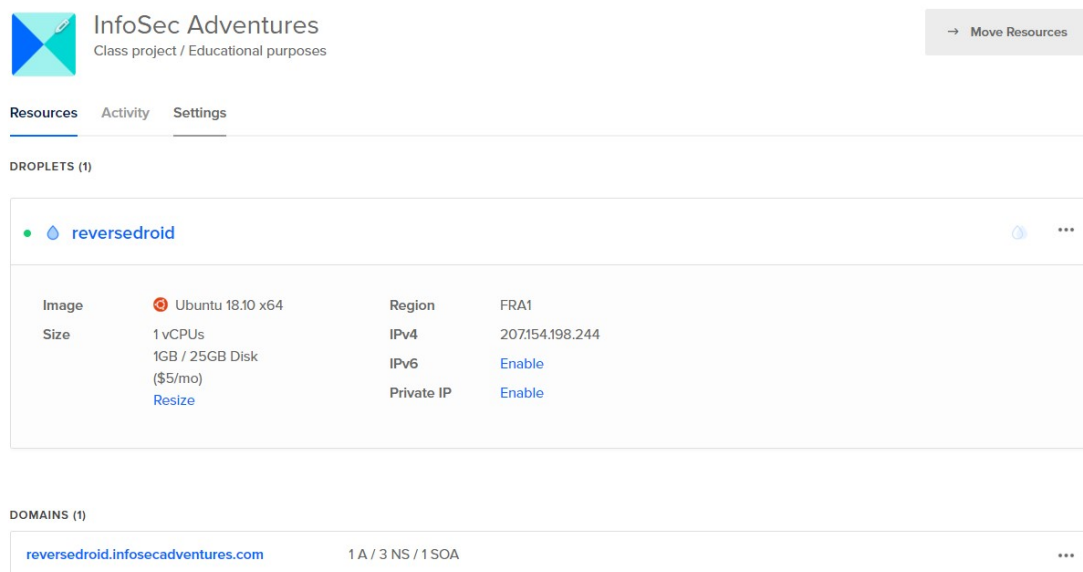
² Application Programming Interface

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 Digital Ocean felhő szolgáltatására esett. Az oldal felületén lehetőségünk van több, úgynevezett *droplet*-et létrehozni, amik nem mások mint virtuális szerverek. Megadhatjuk milyen disztribúciót szeretnénk telepíteni, jelen esetben én az Ubuntu Linux 18.10-es verzióját telepítettem.



2.1. ábra. Droplet a Digital Ocean admin felületén.

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

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

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

2.2. ábra. DNS rekordok a domain beállításában.

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

2.2. Mobil platform választása

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

3. fejezet

Felhasznált technológiák

3.1. Verzió kezelés

Verzió kezelésre a Git-et használtam, ami széleskörben elterjedt a 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 repository-k tárolására. A projekt fejlesztése alatt több gépről dolgoztam és Github a segítségével biztosítani tudtam, hogy mindenhol elérjen a változtatásaimat. Lényegében egy központi szerverként szolgált számomra.

3.2. Folyamatos integrálás

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

A CircleCi integrálható a Github-al, így kényelmesen lehet csatolni a projekt repository-ját. A 3.1 képen látható, hogy minden egyes változtatáskor lefutnak a tesztek. Ezek a tesztek minden alkalommal egy tiszta konténerben vagy virtuális gépen futnak. Az eredményről minden egyes alkalommal értesítést kapunk, így egyből tudhatjuk azt is, ha egy build nem futott le sikeresen.

By project	My branches	All branches	My jobs	All jobs
ReverseDroid-Android	alpha	4 hours ago	SUCCESS #33	add new test user
ReverseDroid	master	1 month ago	SUCCESS #32	fix additional comma
ReverseDroid	master	2 minutes ago	SUCCESS #31	fix for firebase limitations
			SUCCESS #30	Code refactor
			SUCCESS #29	Get username from authorization header
			SUCCESS #28	Add HTTP response codes for missing parameters
			SUCCESS #27	Add file download unit test
			SUCCESS #26	Check for non-existent file
			SUCCESS #25	Add unit test for notification sending
			SUCCESS #24	Code refactor
			SUCCESS #23	

3.1. ábra. Sikeres project buildek a CircleCi felületén.

3.3. Szerveren használt technológiák

Flask

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

SQLite

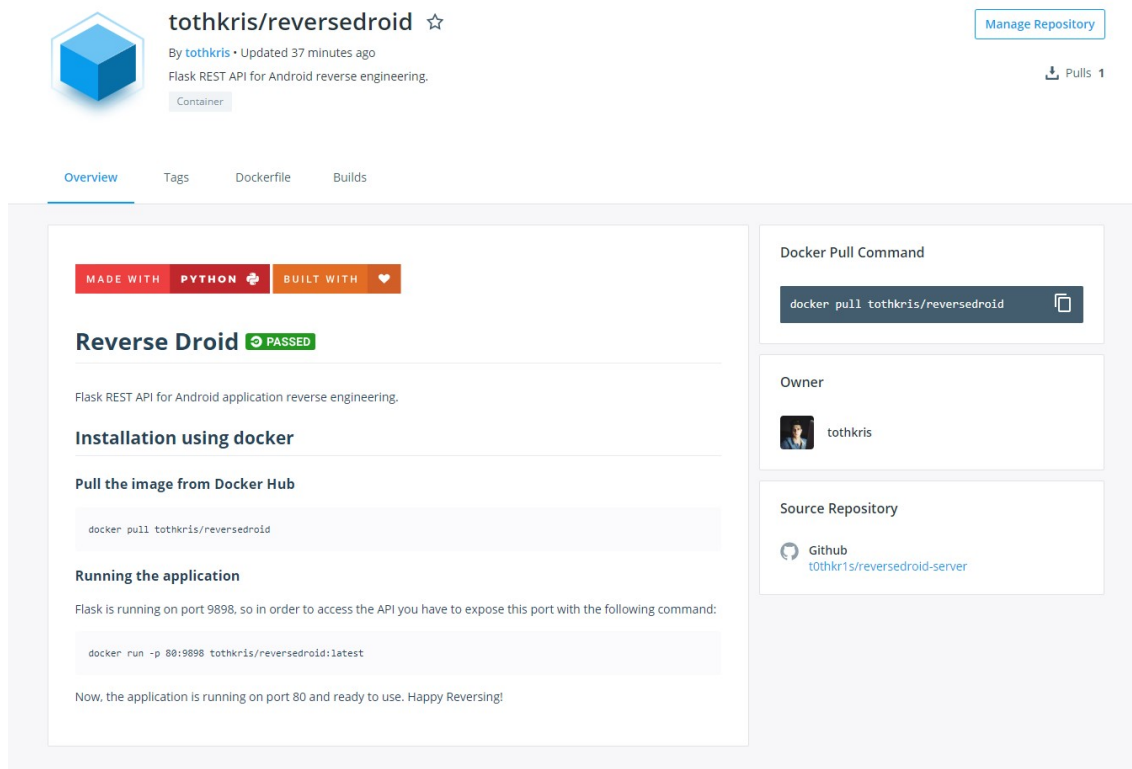
Az SQLite a legtöbbet használt adatbázis motor a világon. Számtalan alkalmazás használja és Android-on is ez az alapértelmezett adatbázis. Az SQLite nyílt forráskódú, így mindenki nyugodtan használhatja.

Pusher Beams

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

Docker

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



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

Tmux

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

3.3.1. jd-cmd

Szükségem volt egy parancssorból használható Java Decompiler-re is.

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

3.3.2. Apktool

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

3.3.3. dex2jar

Android .dex és .class fájlokkal való dolgozáskor kulcs szerepet játszott a *dex2jar*. Jelen esetben nem csak egy programról beszélünk, mert valójában több eszközt foglal

magába. Az Android telepítő fájl kitömörítése után, az alkalmazás forráskódja egy *classes.dex* fájlban található meg. Ez a fájl byte kódot tartalmaz az ART számára és ezt kellett átalakítani .jar formátumba, amivel már a decompiler dolgozni tud.

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

3.3.4. JSON

A JSON¹ 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 XML². Általánosságban strukturált adatok tárolására, továbbítására szolgál. [5]

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.

¹ JavaScript Object Notation

² Extensible Markup Language

3.4.2. OkHttp3

3.4.3. Okio

3.4.4. Pusher Beams

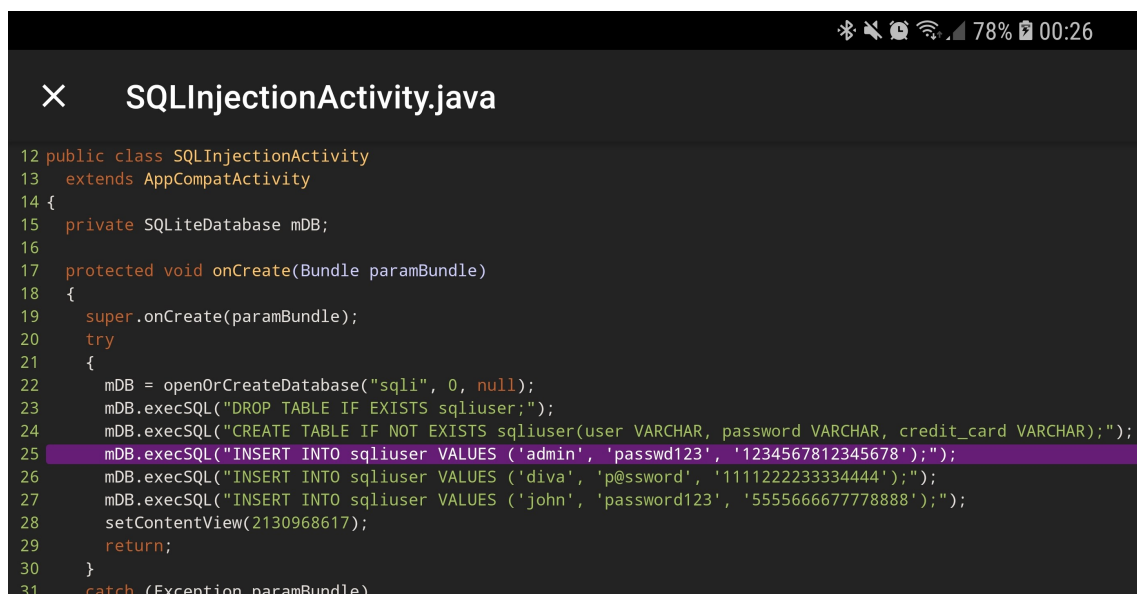
3.4.5. Firebase Messaging

3.4.6. Room

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

3.4.7. CodeView

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



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

3.3. ábra. A vissza fejtett forrás kód megjelenítése.

3.4.8. Espresso

Az Android Studio rendelkezik jó pár funkcióval, amikről már tettem említést. Amit szándékosan kihagytam, az a UI³ 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 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. Állíthatunk elvárásokat és interakciókat anélkül, hogy hozzáférnénk az alkalmazás nézeteihez és activity-jeihez. Ez a struktúra optimalizálja a tesztek futási idejét. [3]

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 3.1 kódrészletben megfigyelhető, hogy egy *@Before* annotációval⁴ 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
7     ActivityTestRule<>(MainActivity . class );
8
9     @Rule
10    public GrantPermissionRule mGrantPermissionRule =
11        GrantPermissionRule . grant (
12            " android . permission . READ _ EXTERNAL _ STORAGE " ,
```

³ User Interface

⁴ Az annotációk a programkód elemeihez rendelhetők (csomagokhoz, típusokhoz, metódusokhoz, attribútumokhoz, konstruktorokhoz, lokális változókhoz), plusz információt hordoznak a Java fordító ill. speciális eszközök számára.[6]

```

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

```

3.1. Listing. Espresso UI teszt az adatok törléséhez.

3.4.9. Material Design 2.0

4. fejezet

Teszteléshez felhasznált eszközök és források

4.1. Sérülékeny Android alkalmazások

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

- Diva
- Sieve
- Pivaa
- Frida

5. fejezet

Megvalósított funkciók

5.1. Bejelentkezés

5.2. Regisztráció

5.3. Kijelentkezés

5.4. Adatok törlése

5.5. Fájl feltöltés

5.6. Fájl letöltés

5.7. Navigáció a fájlrendszerben

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

5.8. Értesítések

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

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

6. fejezet

Továbbfejlesztési lehetőségek

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

7. fejezet

Tapasztalatok

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

Irodalomjegyzék

- [1] ONLINE: Number of mobile phone users worldwide from 2015 to 2020, <https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide>
- [2] ONLINE: Everything you need to build on Android <https://developer.android.com/studio/features.html>
- [3] ONLINE: Create UI tests with Espresso Test Recorder <https://developer.android.com/studio/test/espresso-test-recorder>
- [4] ONLINE: PyCharm Features <https://www.jetbrains.com/pycharm/features/>
- [5] ONLINE: JSON <https://hu.wikipedia.org/wiki/JSON>
- [6] ONLINE: Java annotációk https://hu.wikipedia.org/wiki/Java_annot%C3%A1ci%C3%B3k