

UNIVERSITY OF GRONINGEN

SHORT PROGRAMMING PROJECT

Google Play Store crawler

Dan Plămădeală (s3436624)

Supervisor: Fadi Mohsen

Updated: February 21, 2020



Table of Contents

Introduction	1
Setting up the tool	1
Linux	1
Windows	4
Running the tool	7
Exporting the data	7
Code details	7
Encountered difficulties	8
Limitations in the tool	8
Testing the tool	8
Future development	9
References	10

Introduction

In the context of the Short Programming Project course, I was given the task to find an already existing tool, that after changing slightly and adapting as required, would fulfil the requirements. The requirements of the tool that I worked on were:

- The ability to download .apk files.
- The ability to get the details of an app.

First, I researched the availability of an already existing tool that was at least partly or fully fulfilling the requirements. After reviewing multiple tools as seen in the tool research [1] I started working on the *node-google-play* library [2] to adapt it and create a tool that is to be what I have now.

Setting up the tool

Linux

In order to launch the tool, the user would have to install a specific version of the Node.js runtime environment. To be more precise, version 8.16.2 needs to be installed because this is one of the only versions that doesn't yield an error [3] when trying to connect to the Play Store servers.

This can be done by executing the following steps (for Linux 64 bit):

1. Download the right version from the official Node.js website [4].
2. Create the folder where it is recommended to install Node.js, then extract the contents of the archive to that folder

```
sudo mkdir -p /usr/local/lib/nodejs
sudo tar -xJvf node-v8.16.2-linux-x64.tar.xz -C /usr/local/lib/nodejs
```

3. In order to be able to run Node.js, append the file ~/.profile with the following lines

```
# Node.js
VERSION=v8.16.2
DISTRO=linux-x64
export PATH=/usr/local/lib/nodejs/node-$VERSION-$DISTRO/bin:$PATH
```

4. Refresh the profile

```
. ~/.profile
```

5. Test installation using

```
node -v
```

This should yield the following output:

```
v8.16.2
```

6. Use sudo to symlink the main executables

```
installDir=/usr/local/lib/nodejs/node-v8.16.2-linux-x64

sudo ln -s $installDir/bin/node /usr/bin/node

sudo ln -s $installDir/bin/npm /usr/bin/npm

sudo ln -s $installDir/bin/npx /usr/bin/npx
```

Now that the Node.js is installed, we can clone the repository containing the tool:

```
git clone https://github.com/Cornu11/google_scraper
```

Next, you would have to install the required packages, using **npm**. This can be done by going into the folder with the cloned repository, and executing

```
npm install
```

Now, the most important part is setting up the Google Play Store credentials. This is a bit tricky because you would require an Android phone during the initialization part. You need to set three environment fields, **ANDROID_ID**, **GOOGLE_LOGIN** and **GOOGLE_PASSWORD** that the tool uses for authentication to Google servers. The most straightforward way I could come up with is to put these credentials in a file, that automatically sets them in the environment when needed.

First, you need to install an app on your Android phone, in order to get the Google Service Framework ID (GSF ID). The app I used is Device ID and can be installed from the Play Store [5]. Here is a screenshot of the interface of the app, with the needed ID positioned on the third row:

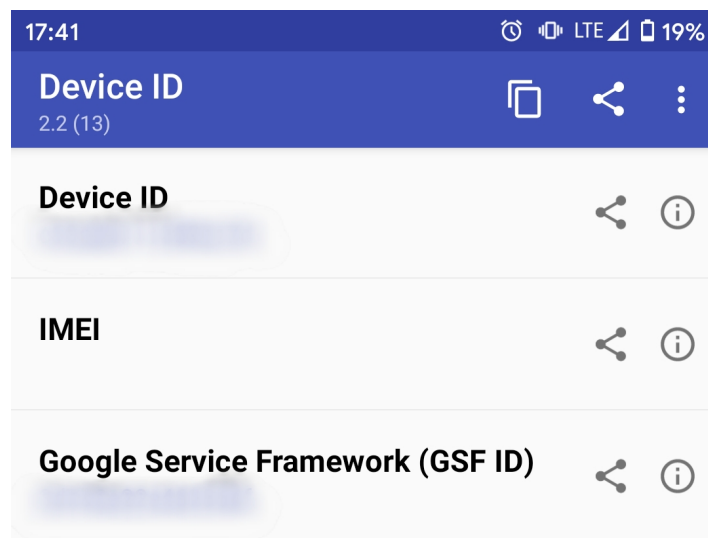


Figure 1: Screenshot of the Device ID app

The next step would be to create a file, for example, **google.rc** somewhere that suits you containing:

```
export ANDROID_ID=abcdefgh12345678
export GOOGLE_LOGIN=your@login.com
export GOOGLE_PASSWORD=yourpassword123
```

Note that the login and the password have to correspond to the ones used on the Android device described by the ANDROID ID in the first line.

Now, that this is done, every time you want to use the tool, you would have to type

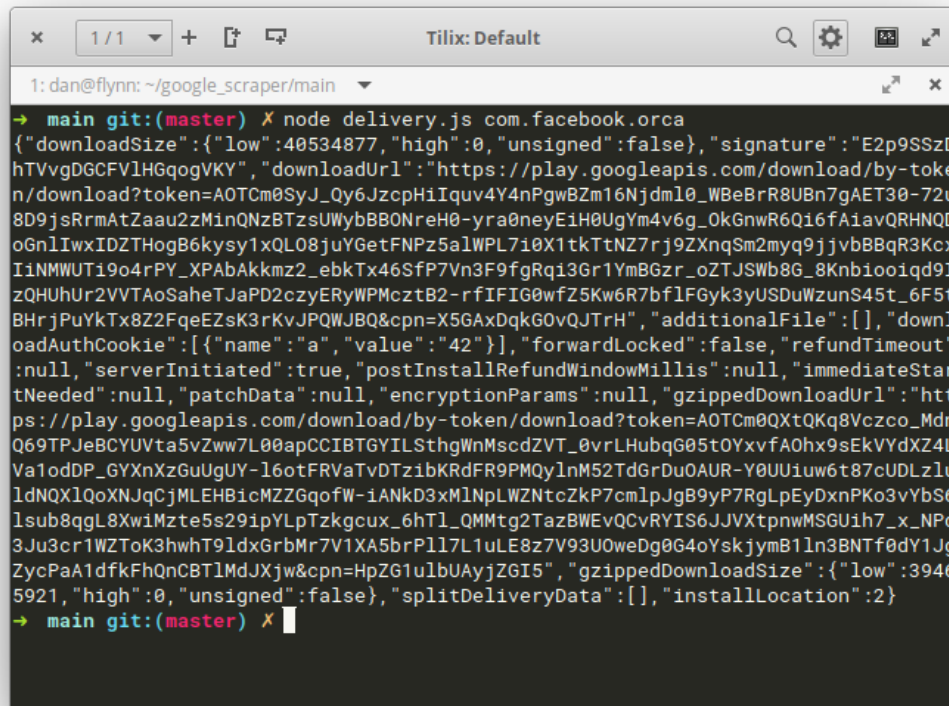
```
source google.rc
```

Or, if you don't want to bother doing this every time, you could just add the command to `~/.bashrc` or any similar file that gets executed at the launch of the terminal session.

To make sure that everything is set up for the tool to work, try running the `delivery.js` script in the `main` folder (`com.facebook.orca` is used just as an example, it works for any appId)

```
node delivery.js com.facebook.orca
```

If you see a compressed JSON output as seen lower, then the tool is successfully retrieving data from the Play Store.



```
1: dan@flynn: ~/google_scraper/main
→ main git:(master) X node delivery.js com.facebook.orca
{"downloadSize":{"low":40534877,"high":0,"unsigned":false},"signature":"E2p9SSzD
hTVvgDGCfVlHGqogVKY","downloadUrl":"https://play.googleapis.com/download/by-tok
n/download?token=A0TCm0SyJ_Qy6JzcpHiIquv4Y4nPgwBZm16Njdm10_WBeBrR8UBn7gAET30-72u
8D9jsRrmAtZaau2zMinQNZBTzsUWybBB0NreH0-yra0neyEiH0UgYm4v6g_0kGnwR6Q16fAiavQRHNQD
oGnlIwxIDZTHogB6kysy1xQL08juYGetFNPz5a1WPL7i0X1tkTtNZ7rj9ZXnqSm2myq9jjvbBBqR3Kcx
IiNMWUTi9o4rPY_XPAbAkkmz2_ebkTx46SfP7Vn3F9fgRqi3Gr1YmBGzr_oZTJSWb8G_8Knbioiqd9I
zQHUhUr2VVTaoSaheTJaPD2czyERYWPMcztB2-rfIFIG0wfZ5Kw6R7bflFGyk3yUSDuWzunS45t_6F5t
BHRjPuYkTx8Z2FqeEZsK3rKvJPQWJBQ&cpn=X5GAxDqkGOvQJTrH","additionalFile":[],"downl
oadAuthCookie":[{"name":"a","value":"42"}],"forwardLocked":false,"refundTimeout"
:null,"serverInitiated":true,"postInstallRefundWindowMillis":null,"immediateStar
tNeeded":null,"patchData":null,"encryptionParams":null,"gzippedDownloadUrl":"htt
ps://play.googleapis.com/download/by-token/download?token=A0TCm0QXtQKq8Vczco_Mdn
Q69TPJeBCYUvta5vZww7L00apCCIBTGYILSthgWnMscdZVT_0vrLHubqG05t0YxvfA0hx9sEkVYdXZ4L
Va1odDP_GYXnXzGuUgUY-l6otFRVaTvDTzibKRdFR9PMQylNM52TdGrDu0AUR-Y0UUiuw6t87cUDLzlu
ldNQXlQoXNjQcJmLEHBicMZZGqofW-iANkD3xM1NpLWZNtcZkp7cm1pJgB9yP7RgLPyDxnPKo3vYbS6
lsub8qgL8XwiMzte5s29ipYLPtZkgcux_6hTl_QMMtg2TazBWEvQCvRYIS6JJVXtpnwMSGUih7_x_NPD
3Ju3cr1WZToK3hwhT9ldxGrbMr7V1XA5brP1l7L1uLE8z7V93U0weDg0G4oYskjymB1ln3BNTf0dY1Jg
ZycPaA1dfkFhQnCBTlMdJXjw&cpn=HpZG1ulbUAYjZ6I5","gzippedDownloadSize":{"low":3946
5921,"high":0,"unsigned":false},"splitDeliveryData":[],"installLocation":2}
→ main git:(master) X
```

Figure 2: Expected result

Considering that the tool is using MongoDB for storing the collected data, we need to install MongoDB. Since there already exists a very good tutorial for any supported platform, I won't cover the installation here. The tutorial can be found on the official MongoDB website [6].

After installing MongoDB, you will have to manually start it at every system restart using

```
sudo systemctl start mongod
```

or, you can optionally ensure that MongoDB will start following a system reboot by issuing the following command

```
sudo systemctl enable mongod
```

Windows

Similar to the Linux OS, the tool doesn't work properly [3] on the latest version of Node.js, thus an older version is required to be installed, more precisely 8.16.2. This can be done by executing the following steps (for Linux 64 bit):

1. Download the right version from the official Node.js website [7].
2. Install Node.js using the downloaded installer. Don't forget to select **"Entire feature will be installed on the hard drive"** of the **"Add to Path"** and **"npm package manager"** option during installation as seen in the screenshot:

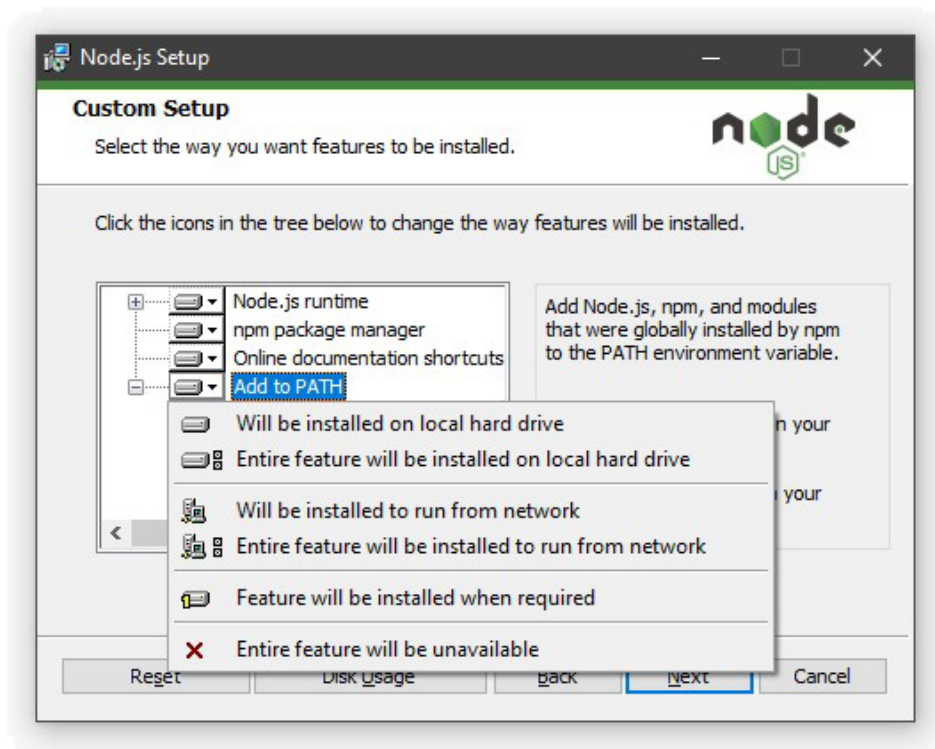


Figure 3: Screenshot of the Installation Window

3. Test installation using the following command in a command line tool of your liking (cmd or PowerShell)

```
node -v
```

This should yield the following output:

```
v8.16.2
```

Now that the Node.js is installed, we can clone the repository containing the tool:

```
git clone https://github.com/Cornul11/google_crawler
```

Next, you would have to install the required packages, using `npm`. This can be done by going into the folder with the cloned repository, and executing

```
npm install
```

Now, the most important part is setting up the Google Play Store credentials. This is a bit tricky because you would require an Android phone during the initialization part. You need to set three environment fields, `ANDROID_ID`, `GOOGLE_LOGIN` and `GOOGLE_PASSWORD` that the tool uses for authentication to Google servers. The most straightforward way I could come up with is to put these credentials in a file, that automatically sets them in the environment when needed.

First, you need to install an app on your Android phone, in order to get the Google Service Framework ID (GSF ID). The app I used is Device ID and can be installed from the Play Store [5]. Here is a screenshot of the interface of the app, with the needed ID positioned on the third row:

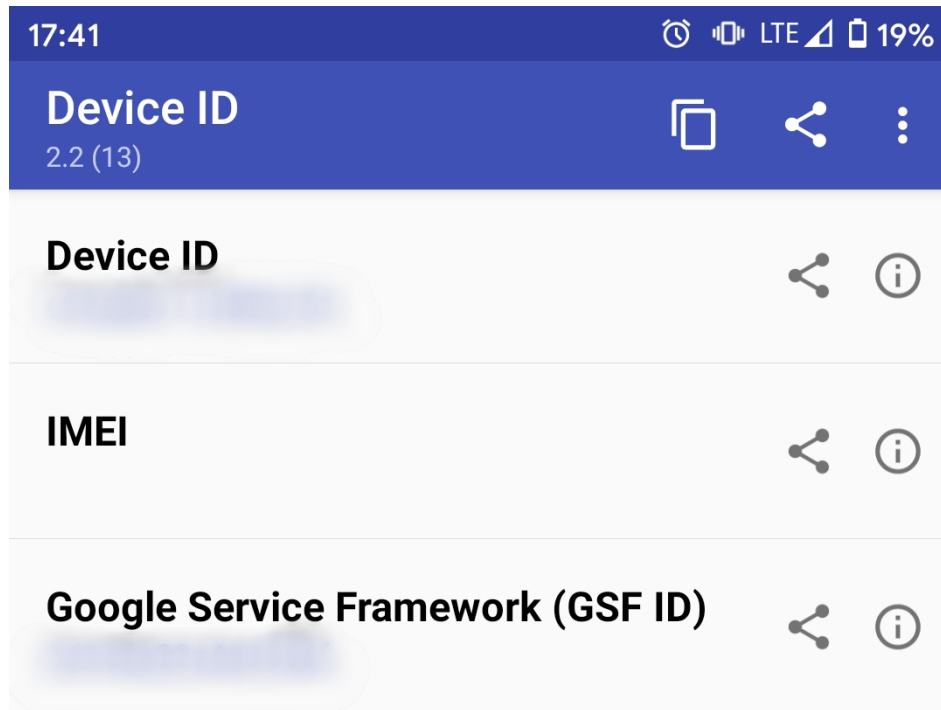


Figure 4: Screenshot of the Device ID app

The next step would be to create a file, for example, `google.bat` somewhere that suits you containing:

```
set ANDROID_ID=abcdefgh12345678
set GOOGLE_LOGIN=your@login.com
set GOOGLE_PASSWORD=yourpassword123
```

Note that using the command `!set!` sets the variables only for this session. You would have to use the `!setx!` command to set the variables permanently.

Note that the login and the password have to correspond to the ones used on the Android device described by the `ANDROID_ID` in the first line.

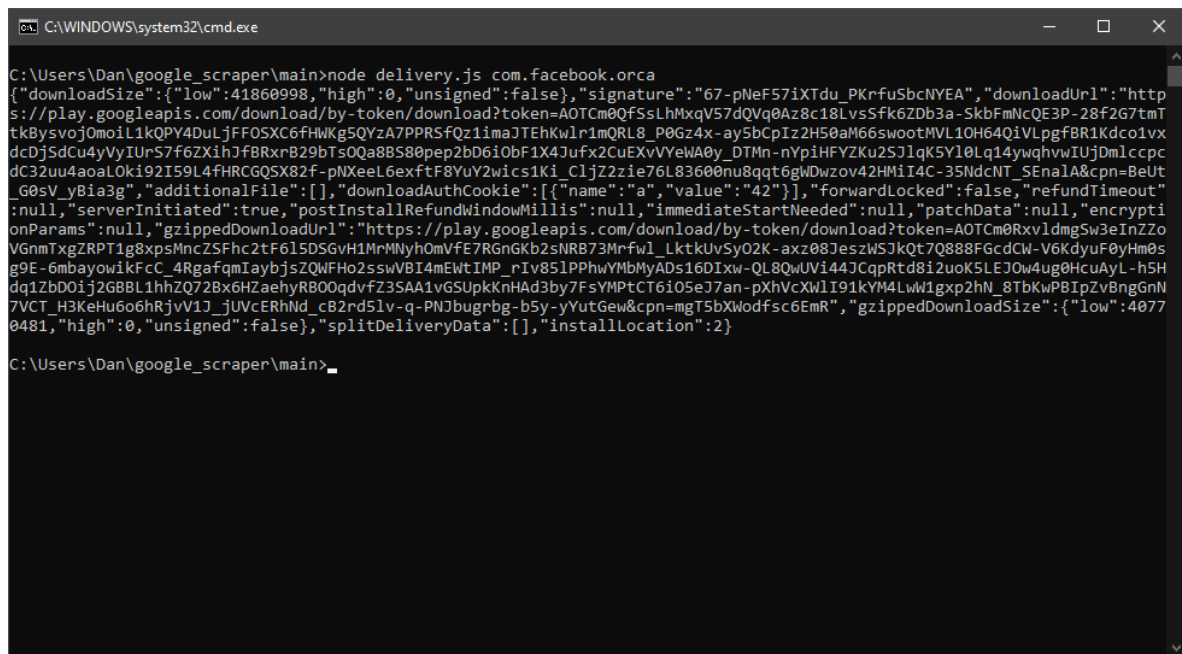
Now, to actually set the variables in the file with the content shown earlier, you would have to simply run the name of the file in the command line.

`google.bat`

To make sure that everything is set up for the tool to work, try running the `delivery.js` script in the `main` folder (`com.facebook.orca` is used just as an example, it works for any appId)

```
node delivery.js com.facebook.orca
```

If you see a compressed JSON output as seen lower, then the tool is successfully retrieving data from the Play Store.



```
C:\WINDOWS\system32\cmd.exe
C:\Users\Dan\google_scraper\main>node delivery.js com.facebook.orca
{"downloadSize":{"low":41860998,"high":0,"unsigned":false},"signature":"67-pNeF57iXTdu_PKrfuSbcNYEA","downloadUrl":"http
s://play.googleapis.com/download/by-token/download?token=A0TCm0QfSsLhMxqV57dQVq0Az8c18LvsSfk6ZDb3a-SkbFmNcQE3P-28f2G7tmT
tkBysvoj0moil1kQPY4DuLjFF0SXc6fHWKg5QYzA7PPRSfQz11maJTEhKwlr1mQRL8_P0Gz4x-ay5bCpIz2H50aM66swootMVL10H64Q1VLpgfB8R1Kdcoivx
dcDjSdCu4yVyIUrS7f6ZXihJfBRxrB29bTs0Qa8B580pep2bD6i0bF1X4Jufx2CuEXvVYewA0y_DTMn-nYpiHFYZKu2S3lqK5Y10Lq14ywhvIUjDmLccpc
dC32uu4aoLOki92I59L4fHRCGQX82f-pNXeeL6exftF8YUy2wics1Ki_CljZzie76L83600nu8qqt6gWdwzov42HMiI4C-35NdcNT_SEnalA&cpn=BeUt
_G0sV_yBia3g","additionalFile":[],"downloadAuthCookie":[{"name":"a","value":"42"}],"forwardLocked":false,"refundTimeout"
:null,"serverInitiated":true,"postInstallRefundWindowMillis":null,"immediateStartNeeded":null,"patchData":null,"encrypti
onParams":null,"gzippedDownloadUrl":"https://play.googleapis.com/download/by-token/download?token=A0TCm0RxxvldmgSw3eInZzo
VGnmTxgZRPT1g8xpsMncZSFhc2tF615DSGvH1MrMnyhOmVfE7RGnGkb2sNRB73Mrfw1_LktkUvSy02K-axz08JeszWSJkQt7Q888FGcdCW-V6KdyuF0yHm0s
g9E-6mbayowikFcC_4RgafqmIaybjsZQWfHo2sswVBI4mEwtIMP_rIv851PPhwYMBMyADs16DIxw-QL8QwUVi44JCqpRtd8i2uoK5LEJOw4ug0HcuAyL-h5H
dq1ZbD0iJ2GBBL1hhZ072Bx6HzaehyR80OqdvfZ3SAA1vGSUpkKnHAd3by7FsYMPtCT6i05eJ7an-pXhVcXwI91kYm4LwW1gxp2hN_8TbKwPBIPzv8ngGnN
7VCT_H3KeHu6o6hRjvV1J_jUVcERhNd_c82rd5lv-q-PNJbugrbg-b5y-yYutGew&cpn=mgT5bXwodfsc6EmR","gzippedDownloadSize":{"low":4077
0481,"high":0,"unsigned":false},"splitDeliveryData":[],"installLocation":2}
```

Figure 5: Expected result

Considering that the tool is using MongoDB for storing the collected data, we need to install MongoDB. Since there already exists a very good tutorial for any supported platform, I won't cover the installation here. The tutorial can be found on the official MongoDB website [8].

After you have installed MongoDB, it will automatically start as a service in the background, thus running the crawler will store the data in the database without any problems.

Running the tool

Running the tool is pretty straightforward, you just have to include the app IDs of the apps that you want to start with, and then the script will do everything for you. This can be done by opening the `main/main.js` and editing the `appIds` array (Below I will explain why this is the way the program works.). Next, you just have to type

```
node main/main.js
```

and the tool will collect the required information to the database and download the `.apk` files corresponding to the crawled apps.

Exporting the data

In order to export the data, you just have to run the `export.js` program. Since the main requirement was to export data in a `.csv` format, running it without any arguments, or with the first argument `csv` will obviously export the data in a `.csv` format. If you run it with the first argument equal to `json`, it will export the collected data from the database in a JSON format. The latter is more convenient because all the data is easier to operate with, as opposed to the `csv` format, because when exporting in `csv`, some fields will be compact and contain more subfields, thus making the data less pleasant to work with.

Concluding, to export everything from the database in a `.csv` format, you have to run the following command in the `main` folder:

```
node export.js csv
```

Code details

As mentioned earlier, the tool is based upon the `node-google-play` library for Node.js. Using this library, the tool is authenticating to the Play Store servers, requesting the information (called `details` in the code) and is downloading the `.apk` files.

The main operations are happening in the `related.js` file. There, the tool is first, requesting all the related apps (`getRelatedApps(pkg)`) of an app ID, after which it is requesting the information about each one of them (`getInfo(pkg)`) and downloading their `.apk` (`api.download(...)`). Next, it is saving the file to the disk (`res.pipe(fileStream)`) and storing the information to the database (`storeInfo(...)`).

All of the steps described earlier are wrapped in the main file (`main.js`) which gets an input array of app IDs, based on which it calls the `related.js` with every app ID. In other words, let's say we have two app IDs, `com.facebook.orca` and `com.facebook.katana`. The tool will now request the related apps to each one of them. This will yield approximately 20 new apps for each one. Next step will be requesting the information about the first app, and then of the other 20 related apps. At the same time, the `.apk` files will be downloaded.

Everything comprised in this tool was designed in such a way so that it is executed sequentially or in other words, in serial. This was done so as to avoid the Play Store banning/blocking

access to us for making too many requests and getting flagged as a DDoS attack. I should mention that between every request, there is a delay of about 1 second so as to solidify this.

To put it another way, in order to get the tool running, one would have to fill in a list of appIDs in the `main/main.js` file on line 14 in the `arr` array. The tool will next sequentially request information about the apps in the list. Let's say for example that my initialization looks the following way:

```
let arr = ['com.facebook.orca', com.facebook.katana'];
```

Next, the tool will request the information and download `com.facebook.orca`. Furthermore, it will request the related apps of `com.facebook.orca`, returning for this specific appID, 20 new appIDs. Correspondingly it will request the information and download the apk of each one of them. After this is done, it will repeat the process for the next appID in the array, `com.facebook.katana`, which has 20 related apps as well (this is the maximum amount of related apps that the underlying framework can get, probably because of the way the Play Store returns the data). Concluding this part, with only two appIDs as the input, we get the information and the apk file of 42 apps.

Encountered difficulties

At the very beginning, I planned was to create a loop that would endlessly request related apps to one input app ID, and add them to the queue and continue on repeating this until reaching a limit of requests, for example. Soon, after researching the way JavaScript promises [9] and asynchronous tasks [10] work, I came to the conclusion that in theory, this approach would be great, but in reality. it won't work because of the fact that Promises are never instantly returned, thus a loop won't work. Moreover, something close to this was working at one stage during development, but again, because of the way JavaScript is engineered, it is stumbling upon a stack overflow at one point [11], because of its' recursive design.

Another point that made the development process more difficult, was the fact that I have never worked before with Node.js, Promises and/or asynchronous tasks. In addition, the code complexity of the library is very high, and changing it is quite a hurdle.

Limitations in the tool

As mentioned in the previous section, because of the way JavaScript is set up, the tool cannot endlessly just crawl the Play Store without any input. This is why the tool requires a list of app IDs that are used as a starting point in the crawling process. The more app IDs the tool is given at the beginning, the more apps will be crawled in the end.

Testing the tool

After achieving a working state of the tool, and a relatively final functionality, I started testing it. I did this by feeding the tool with a list of 193 appIDs. Considering that the tool is based on getting the related apps of every app and the fact that on average we get 13 related apps from the Play Store, this input should yield in theory, information about something around 2700 and of course their apk file.

The delay between requests was set to 25000 ms and a random value between 0 ms and 5000 ms so as to decrease the chances of the Play Store tagging the tool as a crawler/spam tool. In reality, this approach managed to get the information about 98 apps in about 45 minutes as seen in the excerpt from the MongoDB CLI [12]:

```
> use app_data
switched to db app_data
> db.apps.count()
98
```

After the first ≈ 50 apps the first 6-10 requests were succeeding with the rest of them getting blocked. After 98 requests none of the requests was working anymore. Furthermore, 93 apk files were successfully downloaded, the other 5 were paid apps, thus could not be downloaded.

This happened most likely because of the number of requests the tool was making to the Play Store servers. A solution to this is described in the Further developments section.

Future development

- **Automatization:** a script that would automatically fetch all the app IDs in the database, and feed them as an input to the crawler would be great for the scaling of the dataset.
- **RESTful API:** I thought that having a dataset this big, it would be very convenient to have an API that could consume the dataset and provide a useful interface to sort, categorize and visualize the data.
- **Data deduplication:** at the moment, no checking mechanism avoids inserting the details of an app to the database in the case that it is already in the database so such a protection mechanism would be a notable addition.
- **Containerization:** the tool could be packed into a full-fledged Docker container, which would contain all the needed components (Database, Node.js, etc.) and make sure that everything is properly installed and located at the right position with the right dependencies.
- **Distribution:** As stated in an article [13], a proper way of crawling without encountering difficulties, would be to use multiple distributed instances that would concomitantly crawl under different accounts and different network parameters that would most likely solve the problem related to the Play Store servers blocking the requests from the crawler.

References

- [1] Dan Plămădeală. *Available Google Play Store crawlers/scrapers report*. 2019. URL: <https://docdro.id/jWnK9rh>.
- [2] dweinstein. *Google Play API Tool*. 2019. URL: <https://github.com/dweinstein/node-google-play>.
- [3] orgoldfus. *Login Error CaptchaRequired*. 2019. URL: <https://github.com/dweinstein/node-google-play/issues/107#issuecomment-549806957>.
- [4] Node.js. *Node.js v8.16.2 Linux*. 2019. URL: <https://nodejs.org/download/release/v8.16.2/node-v8.16.2-linux-x64.tar.gz>.
- [5] Evozi. *Device ID*. 2015. URL: <https://play.google.com/store/apps/details?id=com.evozi.deviceid&hl=en>.
- [6] Mongo. *Install MongoDB*. URL: <https://docs.mongodb.com/manual/installation/#mongodb-community-edition-installation-tutorials> (visited on 09/01/2019).
- [7] Node.js. *Node.js v8.16.2 Win x64*. 2019. URL: <https://nodejs.org/dist/v8.16.2/node-v8.16.2-x64.msi>.
- [8] Mongo. *Install MongoDB on Windows*. URL: <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows>.
- [9] Jake Archibald. *JavaScript Promises: an Introduction*. 2020. URL: <https://developers.google.com/web/fundamentals/primers/promises> (visited on 12/01/2020).
- [10] Sukhjinder Arora. *Understanding Asynchronous JavaScript*. 2018. URL: <https://blog.bitsrc.io/understanding-asynchronous-javascript-the-event-loop-74cd408419ff> (visited on 10/01/2020).
- [11] fizis. *JavaScript recursion: Maximum call stack size exceeded*. 2012. URL: <https://stackoverflow.com/questions/9497625/javascript-recursion-maximum-call-stack-size-exceeded>.
- [12] MongoDB. *The mongo Shell*. URL: <https://docs.mongodb.com/manual/mongo/>.
- [13] Michael Nielsen. *How to crawl a quarter billion webpages in 40 hours*. 2012. URL: <http://www.michaelnielsen.org/ddi/how-to-crawl-a-quarter-billion-webpages-in-40-hours> (visited on 10/02/2020).