**Google play crawler**

Student: Job HeersinkStudent number: s3364321

**progress report**

**Instructions**  
**special instructions for you**  
since you may or may not have an android device, you might not be able to get an android id. This is why I’ll let you use mine. The credentials are as followes:

email: playstorecrawler00@gmail.com

ww: IAmAGooglePlayCrawler

androidid: 3fe994589ed3f83d  
  
note that this the androidid is linked to my private phone. I would like to keep this id condifential between the two of us. If you would like to use the program for anything order than grading my program, please retrieve your own android id from an android device.

an example of the parameters to give the program are as follows:

--user playstorecrawler00@gmail.com --passwd IAmAGooglePlayCrawler --androidid 3fe994589ed3f83d --package com.snapchat.android --iterations 1000

**general instructions**  
see README.md

**progress**

**week 1**

The first thing I did was gather information about the best way to approach the problem. I looked into scraping the website itself. But the website doesn’t offer the ability to download the app, only the google play app on android devices would allow that. I first began to look into java code, but soon discovered that all available working code was at least 8-4 years old. I tried fixing this code, giving a complete overhaul. But I kept getting stuck at an authentication error. closer look revealed that is was a 403 error code. Which just means I’m not allowed to do anything.  
  
I moved away from the Akdeniz crawler (<https://github.com/Akdeniz/google-play-crawler>) and looked for a more promising Google play api in java. I saw the yeriomin API (<https://github.com/yeriomin/play-store-api>) that was based on Akdeniz but was last updated a year ago (in contrast to Akdeniz, which has last been updated 7 years ago, this is good news!).

Building and testing went perfect, so i regained hope. But i saw this program still uses the old method of getting credentials, which didn't work with Akdeniz as well. But it passed its test so i didnt thinks much of it. after 4 hours of trying to get a grip on the code i decided to test it with my account. The same error from akdeniz revealed itself. This was a dead end.  
  
I looked around for python googple play official api's, but discovered they use the same method for getting the credentials. Which proved to be unsuccessful.  
  
I read about the newly introduced permissions for apps that use google services. I tried implementing those, but there does not seem to be a permission to collect app data from the playstore. Another dead end.

**Week 2**

After one week of searching I found a python program that successfully implemented authentication. (<https://github.com/egbertbouman/APKfetch>). with a little modification and using the more elaborate proto file from the Akdeniz project it was also able to get app details. Downloading apk’s was still a problem. I soon discovered that downloading only works for apps that have been purchased by the account before. I thus had to create a purchase function. I noticed that some older projects implemented this purchase function, but it didn’t seem to work. I tried to implement the purchase sequence that some api's in the past have implemented. Got an error: Your device is not compatible. This would mean that there is something worng with the android id.

The older program uses a Samsung Galaxy Gio GT-S5660 as a fake device. (Just for reference The galaxy Gio was manufacured with android 2.2 and can support max 2.3. we are at android 9 now, it is clearly outdated). I found another project google play api (that doesnt build) containing a file with specifications for a Samsung Galaxy Tab S3 witch can run on 9. I thought this should be recent enough to make it work. But no, it is still incompatible. I found an app which shows all the detailed info of my SAMSUNG Galaxy A70, which the factory default is with android 9 preinstalled. Also incompatible. Eventually I found out that this was because the function“getting of the android id” was faulty. Entering my own android id (from my Samsung galaxy A70) instead of requesting one from the server made it work with quite some modifications.  
After this addition the program was able to download apk files!

**Week 3**

For implementing the permissions I found a list on the developer site of google: "https://developer.android.com/reference/android/Manifest.permission#constants\_2" which is both up to date and more trustworthy than some file on github. I extracted the information and put it into a csv file by scraping the site.  
I managed to get the formatting correct. There is one problem though. An app can have external permissions (not of the android.permission package but from their own package or a package like com.samsung.android.app.spage.permission or com.spotify.music.permission). ofcourse we wont be able to include ALL permissions like that into one file. So I created a speparate .csv file containing the external permissions.

From this point onward, everything went smoothly.  
Apart from some minor bugs that needed fixen I managed to implement getting the reviews, details, apk files, image files and was able to crawl through 500 apps with only 2 errors.

**Week 5**

after the last meeting I also implemented getting the minimal required android version and category info, since I wasn’t able to do that before, by scraping the website instead of the play store app.  
I did some refactoring and made It run for 1000 apps.

after the program did what he had to do. i needed to figure out the timeouts in between the steps. Running for 500 apps on my 20mb/s wifi did not give me a timeout. Only a few incomplete responses. Trying to request from the server again solved that problem.

But when i tried to run for 1000 apps on my parants 100mbps glass fiber internet connection i started to get my first timeouts, since the speed of the downloads now significantly increased. For example. On my internet it takes 14 seconds to download snapchat. On my parents internet it takes 3 seconds.

First i noticed that after 300 apps, i got a server busy response when trying to get the download link. This server busy response would consisntently last for 20 minutes (tested 3 times). I concluded that this meant that i was requesting to much and was thus on a timeout. I made it so that the program would wait a maximum of 50 minutes. with 5 tests in between if the server is free or not before proceeding to the next app. Usually the server timeout would end after the first 2 tries. (the server busy error occurs for all apps. moving on to the next would not solve the problem.)

I finally made it run on my network (20mbps) for 1000 apps: no server timeouts and the overall program ran for 5 and a half hours. While the previous run at my parents (100mbps) ran for 3 and a half hours including a few server timeouts lasting a total of 56 minutes. This proves that even with the server timeouts, but a faster download and more requests per second the program is still faster than no server timeouts, slow download speed an less requests per second. This would imply that my solutions: “waiting around 20 minutes during a timeout instead of waiting 10 seconds after crawling each app” is the faster solution.

As a sidenote the original project lacked in quality and quantity of comments. I improved those quite a bit during the entire duration of the project.

**Detailed description**  
Here I will go file by file giving some insight into how my program approaches the problem.

**googleplaycrawler.py**  
This is the main file.  
Here I will explain the inner workings of my program.

After my program has parsed the arguments, setup the log file and starting timing the program, we begin by trying to authenticate with the provided email, password and android id. From this we get a token, which we can use to authenticate all requests.

After that we load the appinfo.csv file, containing all the previous crawled apps. If NO\_DUPLICATE\_DATA is set to true, the program will only start crawling if the starting package was not in the csv file to prevent duplicate data.

Now the crawling can start. We start of with the given package, getting the details, versioncode and reviews. Then we move on to the downloading. If the DOWNLOAD\_APPS is set to true and the app is free, we perform the purchase. This “purchase” adds the app to our account making it able to download the app. Then we start downloading. After the download has finished, we get the related apps to crawl through later.  
If everything has gone smoothly and no exeptions has been thrown we save the details, reviews and other data into the .csv files.

Then we check if the related apps have been crawled before, to ensure that we won’t enter a loop. We add the related app to the list of visited packages and move on to crawl through this app, repeating the process again.  
  
In case of errors the program does the following:  
Is it a case of “Server busy”, then the program must wait 10 minutes before trying again. Trying a total of 5 times per app before moving on.  
Is it a case of another reason, mostly incomplete response, then the program must wait 1 minute before trying again. Trying a total of 2 times per app before moving on.  
  
  
**apkfetch.proto** and **apkfetch\_pb2.py**.  
the apkfetch.proto file is a file containing the setup of the messages needed to communicate with the google service. Google doesn’t use the general json file to communicate with their service, but their own invention: Protocol Buffers (<https://developers.google.com/protocol-buffers>).  
this .proto file was already provided by the Akdeniz github, and just needed compilation into a python file, the apkfetch\_pb2.py, using the protoc compiler.

**Requirements.txt**

A file containing the Dependencies of the python program. Running pip install -r requirements.txt will install these Python packages.

**LICENSE**  
I included the licence of Egbert Bouman, apkfetch, or in other words the project I based mine on, because the licence said I’d have to: “… The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.”

**resetcsvfiles.py**  
This program, when run, will empty all csv files. You must run this when starting the program over from the same app with NO\_DUPLICATE\_DATA set to true.

**The log folder**  
The log folder contains the log files of previous runs

**The app folder**  
The app folder contains all the downloaded apk files and includes the .csv files within the data folder.

**challenges**  
The most challenging part of this project was definitely getting the authentication to work. I was first stuck with java, trying to find working projects but all of them where significantly outdated. Even a large portion of python projects were. I also spend way to much time looking for solutions instead of moving on to another project or another language. This took about 2 weeks. After I found a project that managed to log me in, things started to move a bit faster. Implementing the purchase was a big challenge, the fact that I found the problem was definitely lucky and the fact that it was solvable by just using my own android id was even luckier

Dealing with different internet speeds was also a challenge. I have an internet speed of 20mbps, the public university has around 50mbps depending on the hour of day and my parents have 100mbps. Because the download of the apk files is the biggest bottleneck, the speed of which that is done is of great influence of the amount of requests per second and thus the frequency of server time-outs. Because it cannot be precisely pinned down when a server is giving the program a time-out, I solved it by making the program wait for 10 minutes each time it gets a server time out, and then trying again for a certain number of times, five in this case. Usually the server timeouts are over in about 20 minutes. So this solution will keep the crawler running, while also keeping the apps skipped to a minimum.  
  
**lessons learned**  
*- do not stick to a familiar language to long if it has proven to be imperfect for the project*  
I wasted around a week trying to find google play crawler java projects and trying to make them work. All the while not even being able to authenticate. I choose to do this mostly due to the fact that java was familiar to me and I never even touched python before.  
But when I decided to move on to python, I noticed how easy the language actually is and how easy it is to get any program running compared to java.  
After a few days of reading python code, I noticed that I should’ve moved on to python sooner.  
  
*- date of last change of a project on github is important*

If there is one pattern, I noticed in my search of projects on github, is that the most unkown but recent projects were more successful at reaching the google play servers than the most known, 8-year-old large projects. It was also a lot easier to implement other/new features to a small barebone, but working, project like APKfetch than a large but faulty project like Akdeniz.