Facebook Profile Sociality Analyzer

TECHNICAL DOCUMENT

AUTHORS:

1. **INTRODUCTION**

This application aims to collect and analyze the reactions of any user’s friends…

1. **FACEBOOK DATA STRUCTURE AND RETREIVING MECHANISM**
   1. **Data retrieving mechanism**

In this application, we use Graph API (version 2.11) supported by Facebook to retrieve data. Facebook (FB) allows applications to retrieve a personal profile or a page with limited data fields. However, applications first must get the user token from the user profile. We obtained this by design a login webpage, registered as FB App. By this way, every Facebook user could be invited to login through this webpage and accept our requested data fields. After all, we would have that user token and their permissions to retrieve necessary data, in our case, are posts, comments, and reactions.

We will request directly to FB Graph API (https://developers.facebook.com/tools/explorer/) with necessary fields to retrieve the data. However, FB API does not allow developers to retrieve in an all-in-one data. Therefore, we will retrieve data in two (2) phases. (1) All the posts will be retrieved and saved to a file. (2) Using the retrieve posts data to extract all post-id and use that post-id to retrieve other fields (comments and reactions). Fields that are retrieved in this phase will then append together and to each of the post.

* 1. **Data structure**

The user token obtained from the user login and acceptance would be transferred to our Main class to retrieve data. The would-be-retrieved FB data is in JSON format (Fig. 1). There are 2 levels of data, we only take care of the 2nd level that is “data”. The Fig. 1 shows the JSON data retrieve from a posts field.

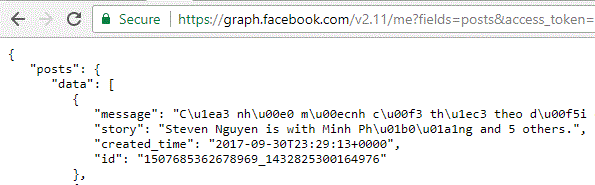


Figure 1. Example of a return of JSON (fields=posts)

As mentioned above, in phase 2, those data would be extracted to have the id and get other fields. Other fields would be appended continue to the “id” in this figure. In short, we will have a list of contents [list] (fig. 2), with each content have a list of message, story, created\_time, id, comments [list], reactions [list].

Figure 2. Data structure created by the application

* 1. **Data storing and encryption**

In the first phase, data would be retrieve from FB Graph API under HTTPS connection. Then, data will be stored under a file with the name as format {filename}.{extension}. The filename is a string obtained from a hash of SHA-256 followed by a Base64 URL-safe-encoding. The hash function uses FB user-id as password and FB user-name as salt. Therefore, this will help to secure the file information that even the developer could not know the end user information. This is a one-way filename encoding which allows an end-user must login into the system with their permission so that the app could get the user-name and user-id to hash (fig. 3)

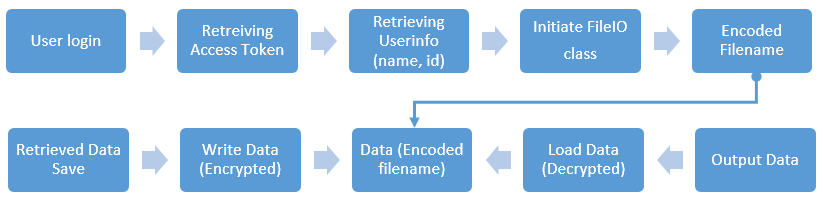


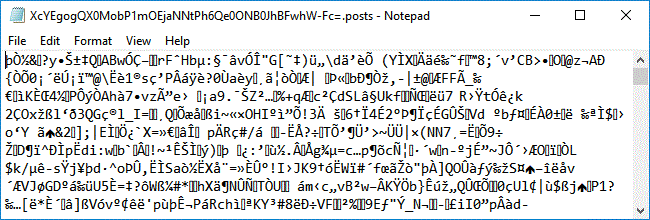
Figure 3. The process of file i/o based on an encoded filename (user-name, id)

For every phase, data will be saved to different file under different extensions (fig. 4). The scope of this is to store separately the posts file and the posts-comments-reactions file, which helps the app to save and read easier, avoid data losing and also for further development (scalable). The extensions are set as “.posts” for posts only file and “.postsNreactz” for posts-comments-reactions file. Those extensions could be set whatever as we want.



Figure 4. Data storing on file system

For privacy and security, we also implemented an encryption and decryption (fig. 5) third-party module and applied inside the functions that take the responsibilities on reading and writing the data. Therefore, data would be encrypted and saved to ensure no one except the end-user could load it. Besides, we also implemented a delete method, which will delete even the encrypted save data on the file system whenever the end-users finish their application uses sessions.

  
Figure 5. An encrypted file opened in Notepad

The module that is used for this is PyCryptodome ([http://pycryptodome.readthedocs.io](http://pycryptodome.readthedocs.io/)). File would be encrypted and decrypted based on AES 128bit key. We provided two options to have the secret key.

(1) The key is a 16-bytes key randomized by the app. Hence, this key would be change from instances of user to user.

(2) The key is the first 16 characters of a combination of FB user-id and user-name. This 16 characters form the max 16-bytes key used in encryption.

By applying two different filename and data encryption, even us could not know which one to decrypted without knowing the user information or if the key is randomized, it could be very secured by the time writing this document. Adding to the security is a file deletion method that would be triggered after finishing data analysis and output to user.

1. **PROGRAMING LANGUAGE**

Python was used to program this. This application was tested under Python version 3.x

Python flask was used to program the web connection to Facebook.

1. **MODULES AND CLASSES**
   1. Built-in modules
   2. Third-party modules
      1. PyCryptodome: [http://pycryptodome.readthedocs.io](http://pycryptodome.readthedocs.io/)

This module is used to encrypt and decrypt the data content, which were retrieved from the FB, before writing to or reading from file system.

* 1. Self-built modules
     1. Core

This module is used to retrieve the JSON data from Facebook, handling errors and retries. Public functions could be triggered as:

+ getUser() *return* a <list> userinfo (name, id)

+ getPosts() *return* a <list> data of all posts from the user

+ getComments() *return* a <list> data of all comments from the user

+ getReactions() *return* a <list> data of all reactions from the user

The fig. 6 demonstrate how all posts would be retrieved from FB.

1. The getPosts() will call the \_getData().
2. This \_getData()
   1. Call the \_prepareURL() function to check which kind of data (posts, comments, reactions) would be retrieved and this function will prepare also a string-type URL to retrieve the JSON data from FB Graph API.
   2. Call the \_getFields() to get the string-type field of which would be retrieved. The JSON would be retrieved as dictionary data type. Therefore, this string is used to know which key to extract the data from that dictionary and append to the final data (<list>)
   3. Call the \_getJSON() after having all necessary data (URL and fields)
      1. There is a while loop inside this function to retry whenever it fails to reach the server. The number of retries could be set in \_\_init\_\_(). There is another error handler inside to catch whenever an error or wrong JSON, lost connection, expired access token… happens
   4. Call the \_getNextURL() to get the ‘next’ keyword in the retrieved dict-type JSON from facebook if there is any other next page because FB will not return all data at once.
   5. Starting a loop with the \_getNextURL() condition, each time, data is retrieved and append to the final data
   6. Call \_checkError() to printout error (if any)
   7. In case the data is None, return an empty list [], otherwise, return data

Figure 6. Process of retrieving Posts

After the posts are retrieved, as mentioned, post id inside the posts data would be extracted to retrieved other data (comments, reactions). This process is almost the same as the process above, except that, the getComments or getReactions will call \_getSubData(). This \_getSubData() holds a <for loop> that run the \_getData() in the loop.

This happens because for any comments or reactions field of a user posts, there could be one or another sub-dictionary containing more than one reactions or comments. Therefore, for every post id, this will make a loop of \_getData(), in which this \_getData() will get all comments or reactions. Everytime the \_getData() returns result, \_getSubData() would append into the parent data (as in fig. 2). A separate loop like this helps to control things easier instead of integrating everything into \_getData().

* + 1. FileIO

This module is used to handle file reading and writing. This file also ensures that data would be saved and open in UTF-8 encoding. Filename and data content would be encrypted and decrypted by this class. Public methods/functions could be triggered:

+ writeToFile() encrypting the file and writing the file into the file system

+ readFromFile() *return* a <list> decrypted data content loading from file system

+ deleteFile() auto delete all files relating to a user

+ getFilePath() *return* a <str> path of the file without extension

* + 1. Main

This module implements other modules (Core, FileIO) to retrieve, save and load data from the internet or from the saved data files.