Documentation of framework modules from: “Automated data population using case-like scenarios for training and validation in mobile forensics”

1. Documentation of the scripts

In order to be able to execute all the scripts correctly, several documentations were made. Every documentation consists of a short description about the use of the respective script. It also contains the usage of other scripts, preconditions, functionality and additional information is given. Please read every documentation carefully and make sure that the preconditions are adhered to.
First, activate the developer options and USB debugging on every smartphone. Then, please install applications needed onto the devices (chapter '1.1 Applications'). Next, set up all the tokens (chapter '1.2 Setup all tokens'). Last, you should load the contacts onto the devices (chapter '1.3 Contacts'). Then, feel free to use all the other scripts.

1.1 Applications

The program 'apps_main.py' installs standard applications on every Android device connected to the PC. This script also installs predefined individual apps to the smartphones. But this is optional.
The devices do have to be connected to the PC.

'apps_main.py' uses the following scripts:

- 'copy_apps_to_folder.py'
- 'install_apks.py'

Preconditions:

Before starting the program, all the applications needed, have to be manually downloaded from the internet as APK-files. The standard apps, that should be installed on every phone, have to be stored in:

'~/formobile/datasets/apps/standard/'

The individual apps, that not every phone gets must be stored in:

'~/formobile/datasets/apps/individual_apps/'

Individual apps are optional and the number of these needs to be defined in the 'config.json' file.

Functionality of the program:

The program first builds up connections to the devices via Android Debug Bridge (ADB). After successfully connected, 'install_apks.py' installs every standard application to all the devices.
The next step is to determine, if devices need individual apps. If not, the program ends. Otherwise, a directory for every device is created, where all the application files are copied to by 'copy_apps_to_folder.py'. Then the individual apps get installed.
To fasten up the downloading speed of applications, threading was implemented.

1.2 Setup all tokens

The program ‘setup.py‘ sets up all tokens needed for the scripts. Tokens for Gmail, Google Calendar and WhatsApp are claimed. It is recommended to execute this program as one of the very first scripts. If the old tokens are not expired yet, the program just finishes with exit code 0. Otherwise, the tokens are claimed one by one. Therefore, it is important to claim the tokens in the exact same sequence, as the persons are declared in the ‘config.json’ file. If problems occur, please delete everything inside the following path and execute the ‘setup.py’ script again:

‘~/formobile/Token/’

The devices do not have to be connected to the PC.

‘setup.py‘ uses the following scripts:

- ‘setup_mail.py’
- ‘cal_setup.py’
- ‘whatsapp_setup.py’

Functionality of the program:

First, the Google Calendar token is claimed by the program for every account by ‘cal_setup.py’. An authentication screen pops up and you need to login. Do this for every account in the correct order, like defined in ‘config.json’.
Next, the Google Mail token is stored on the PC by the ‘setup_mail.py’ script. For claiming it, you need to do the same steps as before.
The last step is to get the WhatsApp security tokens with ‘whatsapp_setup.py’. Therefore, a QR code pops up in the browser. Scan this QR code with WhatsApp on the appropriate smartphone. The script automatically checks if the right person was chosen.

1.3 Contacts

The program ‘contacts_main.py‘ generates a vCard (version 3.0) from the ‘vcard_data.csv’ file and loads it onto the Android device. A vCard is a file format standard for electronic business cards. They can contain name and address information, phone numbers, e-mail addresses, URLs, logos and photographs.
The devices do have to be connected to the PC.

‘contacts_main.py‘ uses the following scripts:

- ‘v_card.py’
- ‘delete_contacts.py’
- ‘send_contacts.py’

Preconditions:

Before executing the program, the file ‘vcard_data.csv’ must be filled manually with the needed data. The format of the data must follow the standard:

’first_names, last_names, birthday, orgs, phone_h, phone_w, street, plz, city, email, picture‘

first_names: first name of contact
last_names: last name of contact
birthday: date of birth in the format YYYY-MM-DD
orgs: organization to which contact is belonging
phone_h: telephone number (home)
phone_w: telephone number (work)
street: address information
plz: address information (postcode)
city: address information
email: email address
picture: profile picture of contact (only tested for .png, .jpg and .jpeg)

Functionality of the program:

The program first builds up connections to the devices via ADB and sets the paths to the needed data. After this, it adds all people from the ‘config.json’ file to the ‘vcard_data.csv’. The new file is called ‘vcard_data_with_config.csv’.
Next, the vCard is generated with the script ‘v_card.py’. The finished vCard is ‘vcards.vcf’.
Now, the contacts are getting deleted with ‘delete_contacts.py’ to avoid duplicate contacts. The vCard can now be loaded onto the device with ‘send_contacts.py’.
To fasten up the sending process of contacts, threading was implemented.

1.4 Pictures

The program ‘main_pictures.py‘ fills the Android device with a predefined number of pictures. The photographs are from the Stanford Visual Genome Dataset and are free to use. The full data set consists of more than 108.000 individual photos and needs approximately just over 15 GB of storage. It can be found under:

‘~/VG_100K/’

In order to use only pictures without any location data, we removed the photos with metadata. Therefore, we used the script ‘remove_metadata.py’. It also reduced the size of the data set to under 5 GB. The full metadata-free data set is stored under the path:

‘~/formobile/datasets/pictures/without_meta/’

Please load only pictures without metadata onto the smartphones.
The devices do have to be connected to the PC.

‘main_pictures.py‘ uses the following scripts:

- ‘copy_pictures_to_folder.py’
- ‘modify_timestamps.py’

- ‘send_pictures.py’

Preconditions:

Under the directory

‘~/formobile/datasets/pictures/without_meta/’

you can find all the images without metadata. This is our original source. The program will copy picture by picture to the destination directory and afterwards delete it from the data set automatically. Therefore, you should **NOT** use the ***‘without_meta’*** folder. Just make a copy of it in the same directory and name it **‘without_meta_copy’**.
If the **‘without_meta_copy’** folder doesn’t contain enough pictures for your purposes, please delete this folder and create a fresh copy of ***‘without_meta’*** under the name **‘without_meta_copy’.**

Functionality of the program:

First, the program builds up connections to the devices via ADB. Then, the program copies the predefined number of images from the source directory to the destination folders of each device with ‘copy_pictures_to_folder.py’. It also removes every copied image from the source, in order to avoid duplicate pictures on different devices. Therefore, it is important to **NOT** use the ‘copy_pictures_to_folder.py’ script with the original images data set. To adjust the amount of pictures, change the ‘config.json’ file.
Afterwards, the timestamps are getting modified and randomized through ‘modify_timestamps.py’ to the given start and end dates defined in the ‘config.json’ file. The naming convention of photos for every smartphone is read from the ‘config.json’ file.
Finally, ‘send_pictures.py’ sends every image inside following path to the respective device:

'~/formobile/datasets/pictures/gallery/<serial number>'

To fasten up the sending process of photos, threading was implemented.

1.5 Phone calls

The program ‘phonecall_main.py‘ makes an automated call between two devices with a duration of just over 12 seconds. At the end, one of the two parties hangs up. This is randomized.
The two devices do have to be connected to the PC.

‘phonecall_main.py‘ uses the following script:

- ‘simulate_phone_call.py’

Preconditions:

The caller and the called person need to be defined in the script.

Functionality of the program:

First, a connection to the devices is made via ADB. Afterwards, the caller and the called person are defined and the phone numbers are set like in the ‘config.json’ file. Now, the ‘simulate_phone_call.py’ script is executed. This starts the automated call and ends it automatically after circa twelve seconds.

1.6 Browser

The program ‘chrome_main.py‘ populates the browser history of every smartphone.
The devices do have to be connected to the PC.

‘chrome_main.py‘ uses the following script:

- ‘populate.py’

Preconditions:

A list of possible websites and Google searches needs to be created. Therefore, create a text file with every link on a new line. It is important to declare the protocol ‘http’ and the ‘www’, for example:

‘http://www.google.com’

If you wish to use Google searches, you should add your keywords after the ‘=’ with ‘+’ separated. For example, searching for the PyCharm IDE looks like this:

‘http://www.google.com/search?q=pycharm+ide’

Functionality of the program:

First, a connection to the devices is made via ADB. Afterwards, you can decide to open every link in the text file or just a specified number of random links. After that, the links are opened one by one on the smartphone and the program waits three seconds, so that Chrome can log the visited website to your browsing history.
To fasten up the opening process of links, threading was implemented.

1.7 GPS Spoofing

The program ‘spoofing_main.py‘ manipulates the devices location through the Appium Settings app. It is possible to send latitude and longitude GPS information to the device to mock the location.
The devices do have to be connected to the PC.

‘spoofing_main.py‘ uses the following scripts:

- ‘enjoying.py’
- ‘working.py’

- ‘weekend.py’

Preconditions:

The enjoying, working and weekend hours and the location for every time period needs to be set for every person in the ‘config.json’. ‘Enjoying’ means the spare time after work.
You also need to install the Appium Settings app onto the devices. Therefore, connect the smartphones to the PC and start ‘download_delete.py’. Make sure that the argument of the ‘ download_delete’ function is set to ‘download’. If you no longer need to spoof locations, you can delete the Appium settings app from your devices. Make sure that the argument of the ‘ download_delete’ function is set to ‘delete’.

Functionality of the program:

Make sure that the argument of the ‘location_spoofer’ function is set to ‘start’. Now, connections to the devices are made via ADB. Then, Appium is loaded into the RAM of every smartphone and the GPS spoofing starts automatically. Google Maps opens up automatically to log the location. Every 30 minutes the program restarts in order to hold Appium in the RAM. If you wish to stop GPS spoofing, please stop the program, and make sure that the argument of the ‘location_spoofer’ function is set to ‘stop’. Now you can restart the script.
To fasten up the speed of manipulating GPS locations of applications, threading was implemented.

1.8 Reddit data

The program ‘reddit_main.py‘ prepares any given subreddit for our chat purposes on WhatsApp.
The devices do not have to be connected to the PC.

‘reddit_main.py‘ uses the following scripts:

- ‘get_mods.py’
- ‘concat_subreddit_dl_data.py’
- ‘reddit_comments.py’

Preconditions:

Before executing the program, a subreddit needs to be chosen. You can find subreddits here:

’https://www.reddit.com/subreddits/’

After finding an appropriate subreddit, you must write the **exact** name of it into the variable ‘subreddit’ in the script. Important to write it exactly like on Reddit, because it is case sensitive.

Functionality of the program:

First, the module starts to download the specified subreddit (this may take a while) via the Linux shell. Afterwards the downloaded files are concatenated to one main file through ‘concat_subreddit_dl_data.py’.

Now, a JSON list of all the moderators of the given subreddit will be downloaded automatically. Then the moderators are filtered out from the list. The script doing this, is called ‘get_mods.py’.
Finally, the data is getting prepared automatically by ‘reddit_comments.py’. It deletes every comment of the moderators and formats the data to our needs. The finished data set is called ‘group_chats.json’ and is under the following path:

‘~/formobile/datasets/<subreddit>/’

If problems occur, please delete the created folders of your subreddit and retry the whole process.

1.9 Calendar

The program ‘calendar_main.py‘ creates calendar entries like declared in the ‘PERSON_<letter of the person>.csv’ under the following path:

‘~/formobile/google_calendar/events/’

In order to be able to manage calendar entries, manually created Google tokens are needed for every account. These tokens expire 2 weeks after creation. So, they regularly need to be claimed.
The devices do not have to be connected to the PC.

‘calendar_main.py‘ uses the following scripts:

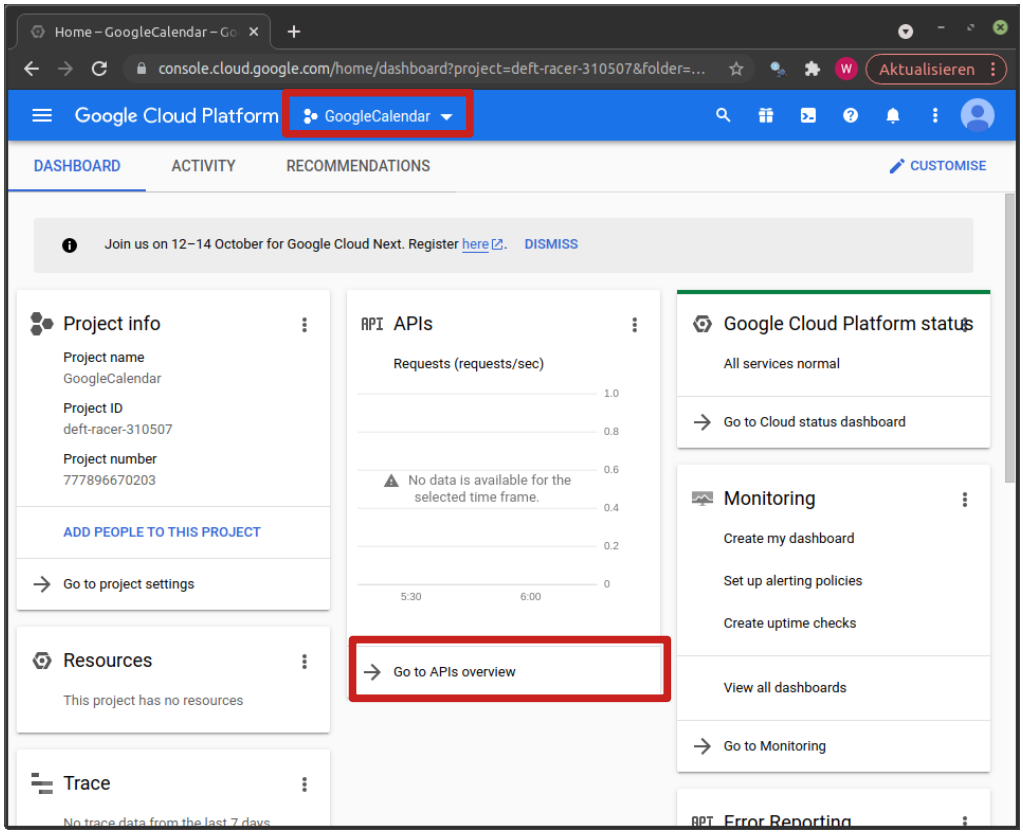
- ‘write_events.py’
- ‘clear_calendar.py’
- ‘create_event.py’

Preconditions:

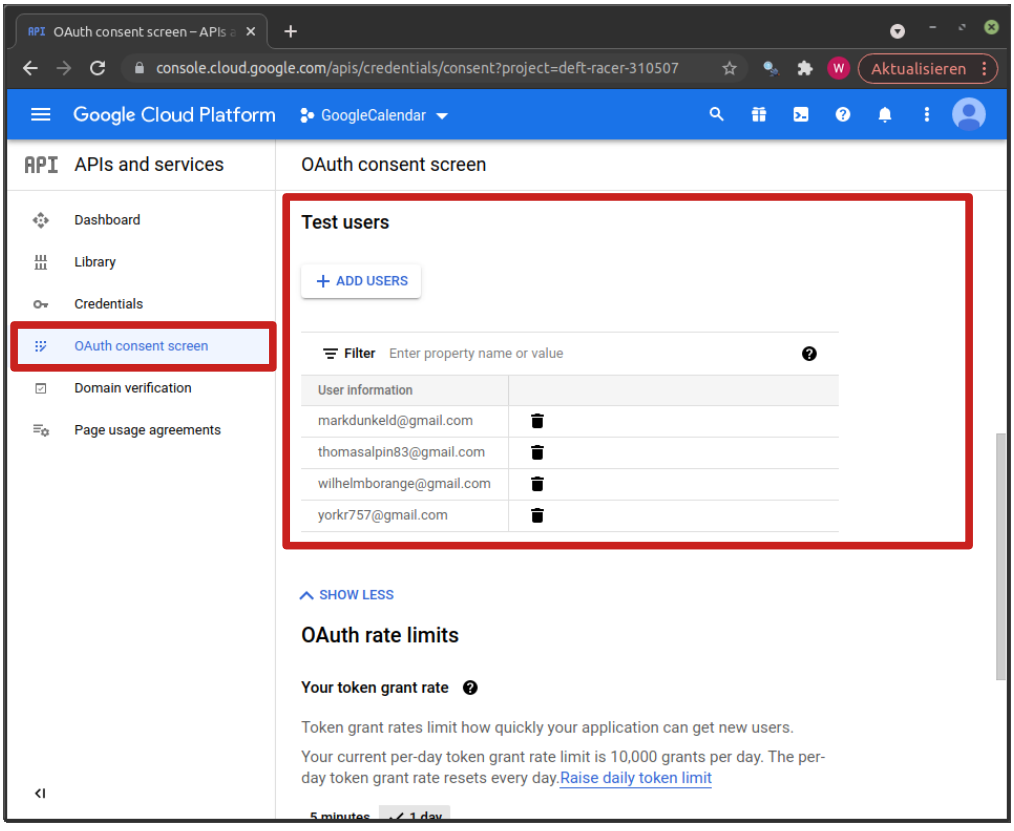
Go to the Google Cloud Platform and log in with Wilhelm B. Orange’s account:

‘https://console.cloud.google.com/’

First, it is important to select the so called ‘GoogleCalendar’ project. Then click on ‘Go to APIs overview’.



Now, go to the menu ‘OAuth consent screen’. Here you should check, if every declared person in the ‘config.json’ file is listed under the section ‘Test users’. If not, you have to add the person(s) manually.



Now, you are able to use the Calendar API.
Nice to know: The ‘credentials.json’ can be downloaded under the menu ‘Credentials’ but is already stored on this machine. Therefore, this step is not needed.

Next, the ‘PERSON_<letter of the person>.csv’ files need to be filled with data. The format of the data must follow the standard:

’summary, description, location, start, end, timezone, organizer, attendees, recurrence, how often, specify days’

summary: title of the event
description: further explanation
location: place of meeting
start: start date in the format YYYY-MM-DDTHH:MM:SS (example: 2021-01-01T12:00:00)
end:end date in the format YYYY-MM-DDTHH:MM:SS
timezone: Europe/Berlin
organizer: head of meeting
attendees: participants of the meeting
recurrence: daily, weekly, monthly or yearly
how often: how many times
specify days: MO, TU, WE, TH, FR, SA, SU

The last step is to execute ‘calendar_main.py’. Google tokens need to be claimed. If they are expired or not existent, an authentication screen will pop up. You just need to log in with the persons Google account and the token will automatically be stored in your local storage.

Functionality of the program:

‘calendar_main.py’ executes the ‘populate_calendar’ function. This function sets the paths to the tokens and to the ‘PERSON_<letter of the person>.csv’ files. If you do not want several identical calendar entries, it is recommended to let the old entries be deleted by the program. Therefore, write ‘TRUE’ as the argument when calling the ‘populate_calendar’ function. This calls the script ‘clear_calendar.py’ and removes every entry.
Next, the events from ‘PERSON_<letter of the person>.csv’ are written to the ‘events.csv’ file with the ‘write_events.py’ script.
In the last step, every event is synchronized in Google Calendar over the API with ‘create_event.py’.

1.10 Emails

The program ‘gmail_main.py‘ sends automatic emails over gmail accounts of the participants declared in the script and in the ‘config.json’ file. In order to be able to send emails, manually created Google tokens are needed for every account. These tokens expire 2 weeks after creation. So, they regularly need to be claimed.
The content of every email originates from Reddit. More precisely, the data comes from the subreddit ‘SeriousConversation’ and was already prepared to our needs.
The devices do not have to be connected to the PC.

‘gmail_main.py‘ uses the following scripts:

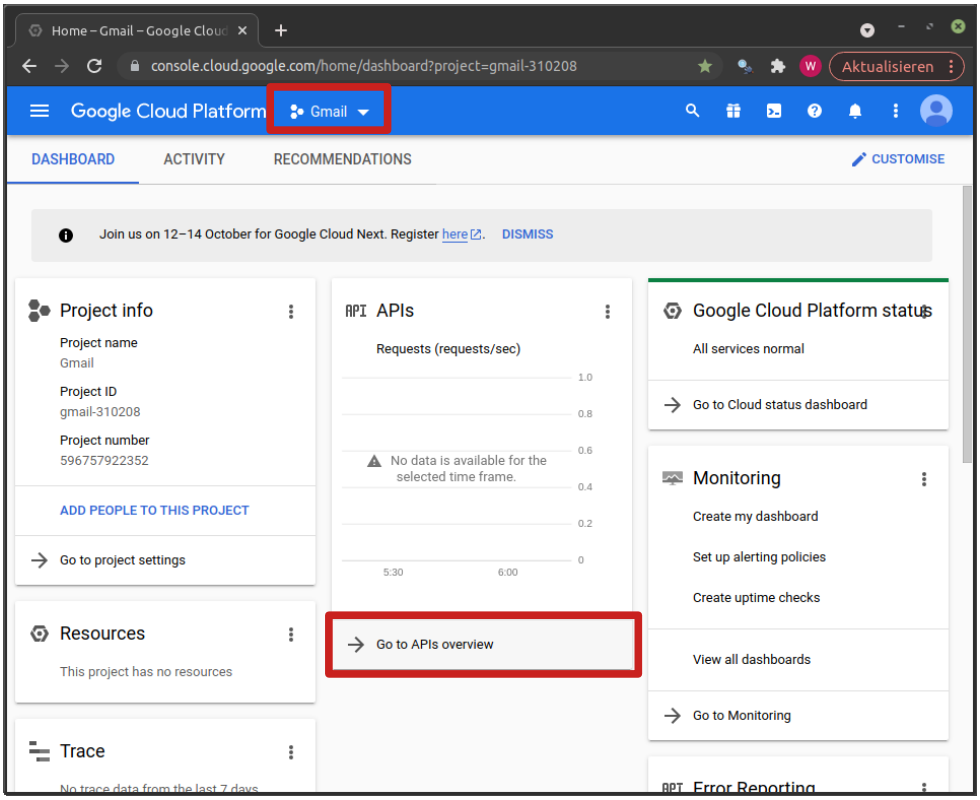
- ‘gmail_account.py’
- ‘get_mods.py’
- ‘reddit_email.py’

Preconditions:

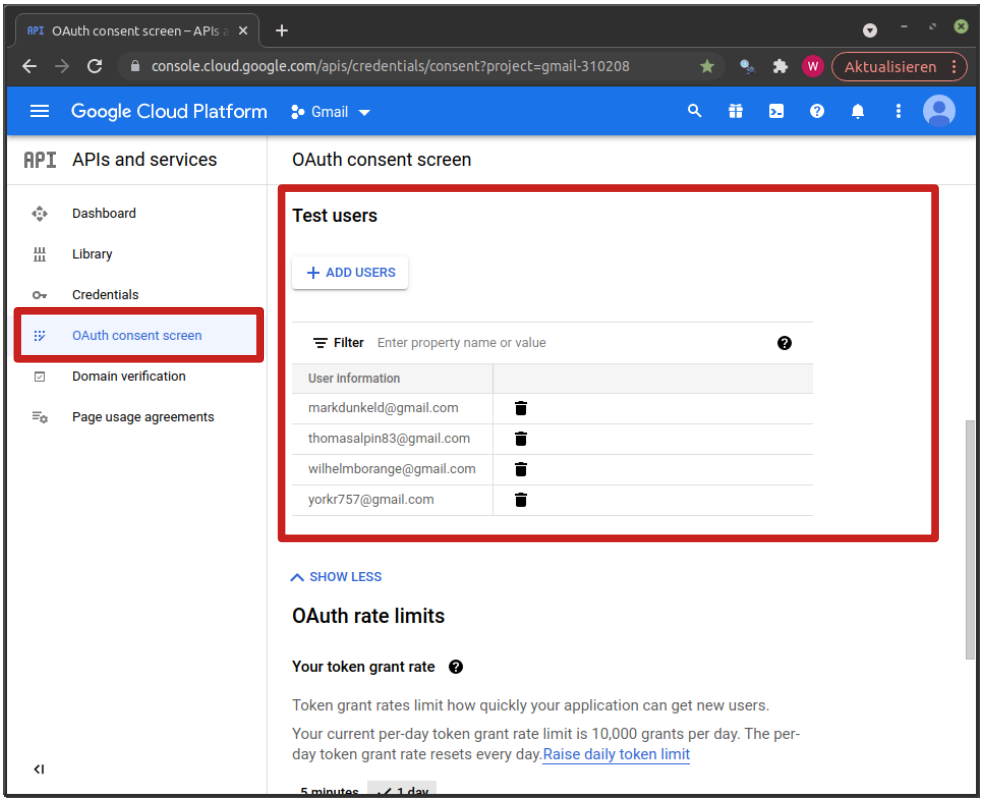
Go to the Google Cloud Platform and log in with Wilhelm B. Orange’s account:

‘https://console.cloud.google.com/’

First, it is important to select the so called ‘Gmail’ project. Then click on ‘Go to APIs overview’.



Now, go to the menu ‘OAuth consent screen’. Here you should check, if every declared person in the ‘config.json’ file is listed under the section ‘Test users’. If not, you have to add the person(s) manually.



Now, you are able to use the Gmail API.
Nice to know: The ‘credentials.json’ can be downloaded under the menu ‘Credentials’ but is already stored on this machine. Therefore, this step is not needed.

The data sent with the emails is already prepared and ready to send. If you wish to use another subreddit instead, please use ‘reddit_main.py’ to download the data and change the paths in ‘gmail_main.py’.

Next, execute ‘gmail_main.py’. Google tokens need to be claimed. If they are expired or not existent, an authentication screen will pop up. You just need to log in with the persons Google account and the token will automatically be stored in your local storage.

Functionality of the program:

First the path to the ‘credentials.json’ file is set. This file is from the Google Cloud Platform and let us use the Gmail API.
Now, the conversation partners are set. Therefore, it is important to name the Gmail token, Gmail address, first name and the signature of every conversation partner.
In the next step, the moderators from the data set are filtered out with the ‘get_mods.py’ script. This enables us to format the email conversations data with ‘reddit_email.py’ to our needs, e. g. by removing comments of deleted posts.
Finally, ‘gmail_account.py’ sends the messages and also replies to emails gotten in the past.

1.11 Finite State Machine & WhatsApp

The program ‘main_fsm.py‘ represents a finite state machine (FSM), needed for realistic automated WhatsApp chats. The FSM determines the current state of the WhatsApp chat bot based on probabilities given in the transition matrix. It can be modified in the ‘config.json’ file.
The devices do not have to be connected to the PC.

‘main_fsm.py‘ uses the following script:

- ‘whatsapp_fsm.py’

Preconditions:

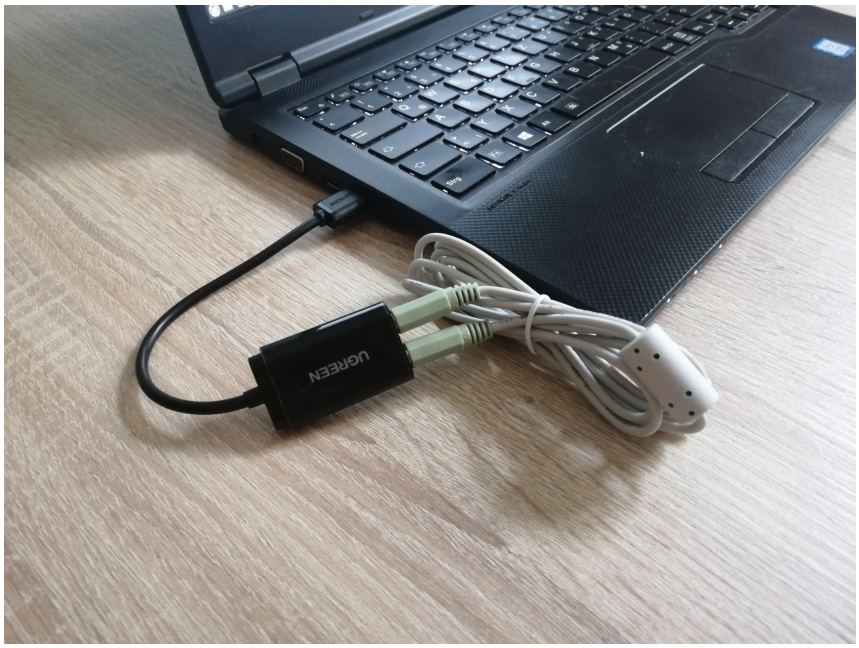
First, the transition tables for every person defined in the ‘config.json’ file should be modified to your needs. Therefore, please use the template under:

‘~/formobile/messenger/whatsapp/modular_fsm_test/transition_matrix.ods’

Save it as a ‘csv’ file for every person.
The data sent in private conversations is from the Amazon Alexa AI group and should not be modified.
The data sent in group chats is already prepared and ready to send. If you wish to use another subreddit instead, please use ‘reddit_main.py’ to download the data. Afterwards, change the paths in ‘definitions.py’ in the variable ‘REDDIT_DATA’. You must also delete the following file:

‘~/formobile/messenger/whatsapp/modular_fsm_test/conversations.json’

The program is also able to send automatic voice messages via a text to speech script. Therefore, before starting the program, you will need a sound card and an AUX cable. Please plug one end of the AUX cable into the input and the other end into the output of the sound card. Plug the sound card in the USB slot of your PC, like in the picture below.



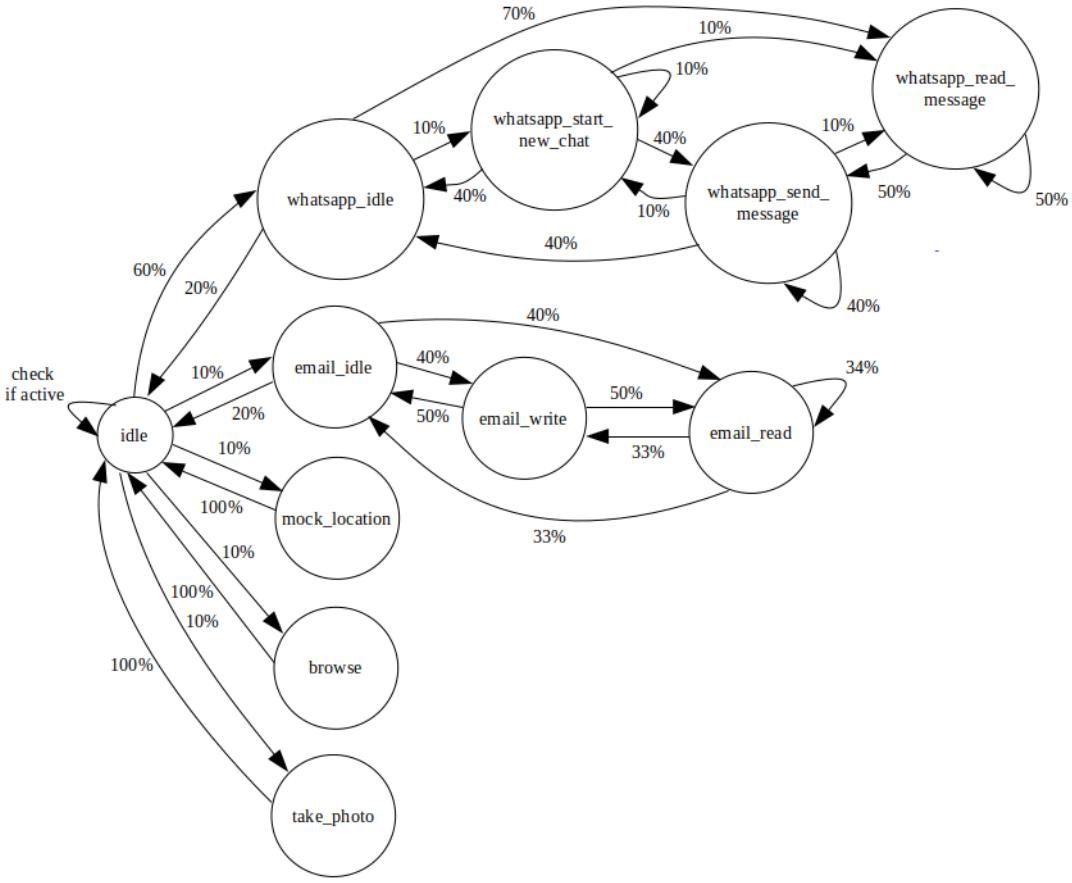
When first starting the program, WhatsApp Web opens and a QR code is displayed. Please scan the QR code with the appropriate device. The security token, which is made by scanning the code, is stored on your local machine. Therefore, no authentication is required for this device in the future. Despite this, the device still needs an active internet connection, so that WhatsApp Web can work properly.

Functionality of the program:

First, the program creates n (= number of persons defined in 'config.json') objects from the class 'Person' and sets the needed attributes.
Then, the 'start' function is called for every person. Within this function an endless loop calls the 'whatsapp_fsm.py' script. This script decides in which state every person is. Following states are currently possible:

'idle, whatsapp_idle, whatsapp_read_message, whatsapp_start_new_chat, whatsapp_send_message, email_idle, email_read, email_write, mock_location, browse, take_photo'

idle: person waits for command
whatsapp_idle: person is online, but inactive
whatsapp_read_message: marks the message as read ('makes the two arrows blue')
whatsapp_start_new_chat: starts a chat with someone new
whatsapp_send_message: sends a text or voice message
email_idle: person is online, but inactive
email_read: marks the message as read
email_write: sends a text message
mock_location: spoofs the GPS location
browse: creates a browsing history
take_photo: saves a photo in the gallery



Once, a state is defined for a person, the ‘whatsapp_fsm.py’ script automatically does the things needed to be done in every state. For example, in the state ‘send_message’, we expect that a message is sent to a random person. The script also decides automatically, if a normal text message or a voice message is being sent.
To fasten up the conversations and make them more authentic, threading was implemented.