

Python Project - WhatsApp Automation System

Imran Hakim Roslan
imrh@protonmail.com

February 20, 2021

1 Introduction

This project applies Python programming and takes the advantage of WhatsApp Web to implement it as an automation interface for easy interaction and user friendly. PyQt5 is used as graphical user interface for the application. A selenium Python library interconnects the Python scripting program with WhatsApp Web.

The Python automation script reads the latest message from a given contact number and performs automation job accordingly.

2 Project Goals

The main goals of the project are listed below:

1. A platform for the Python script to interconnect with WhatsApp Web.
2. Extra WhatsApp operations functionality that user can put to use which are not available on the WhatsApp platform itself.
3. An easy one click button to start the automation scripting process.
4. A Python script to read incoming messages from a user and starts translating the user input commands.
5. A defined set of commands that sit behind the application the for the Python script to act on by reading user commands and activates those commands.

3 Methods and Procedures

3.1 Goal 1: A Graphical User Interface (GUI) Application Platform

To create a GUI for the application, PyQt5 has been used for its plenty of features available. The GUI is made of five sections to categorize different functionality of the application. Each numbered sections in the Figure 1 and discussed below.

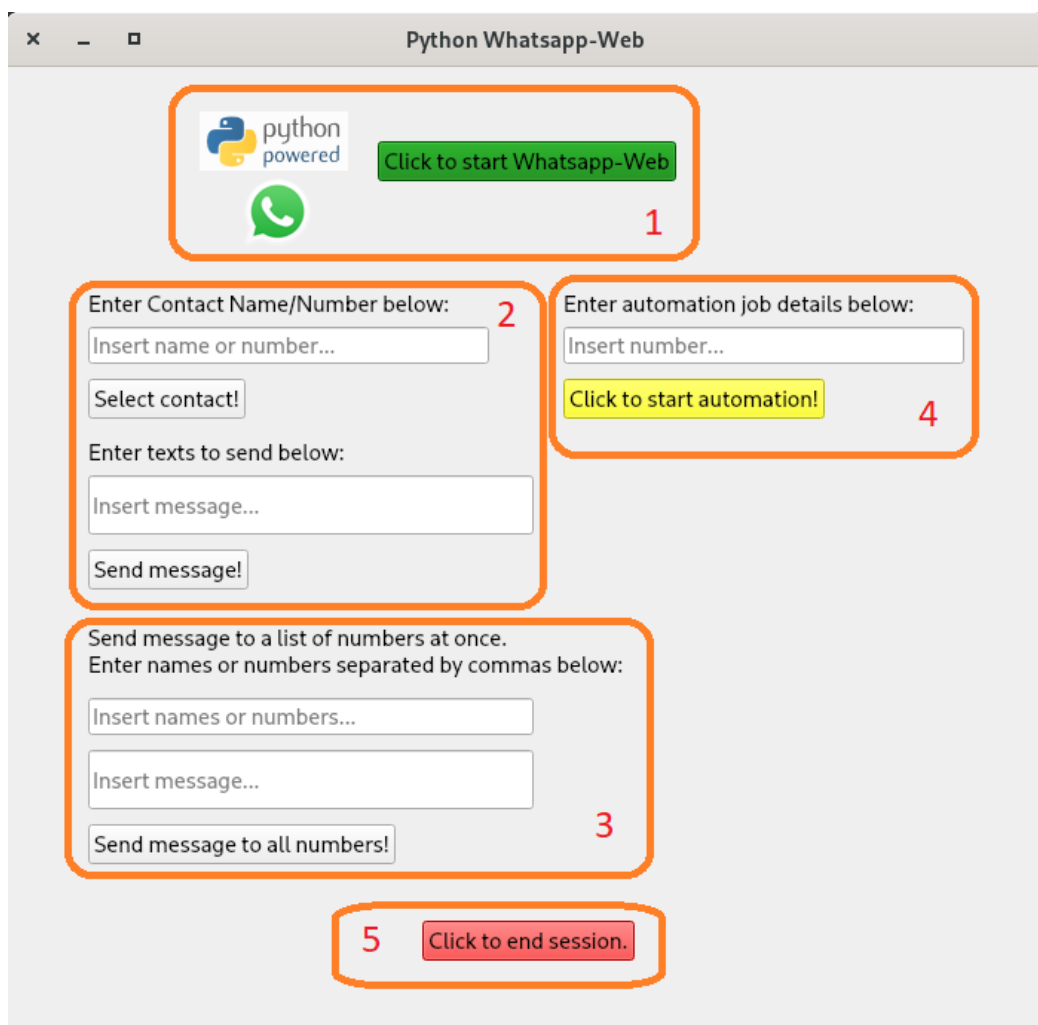
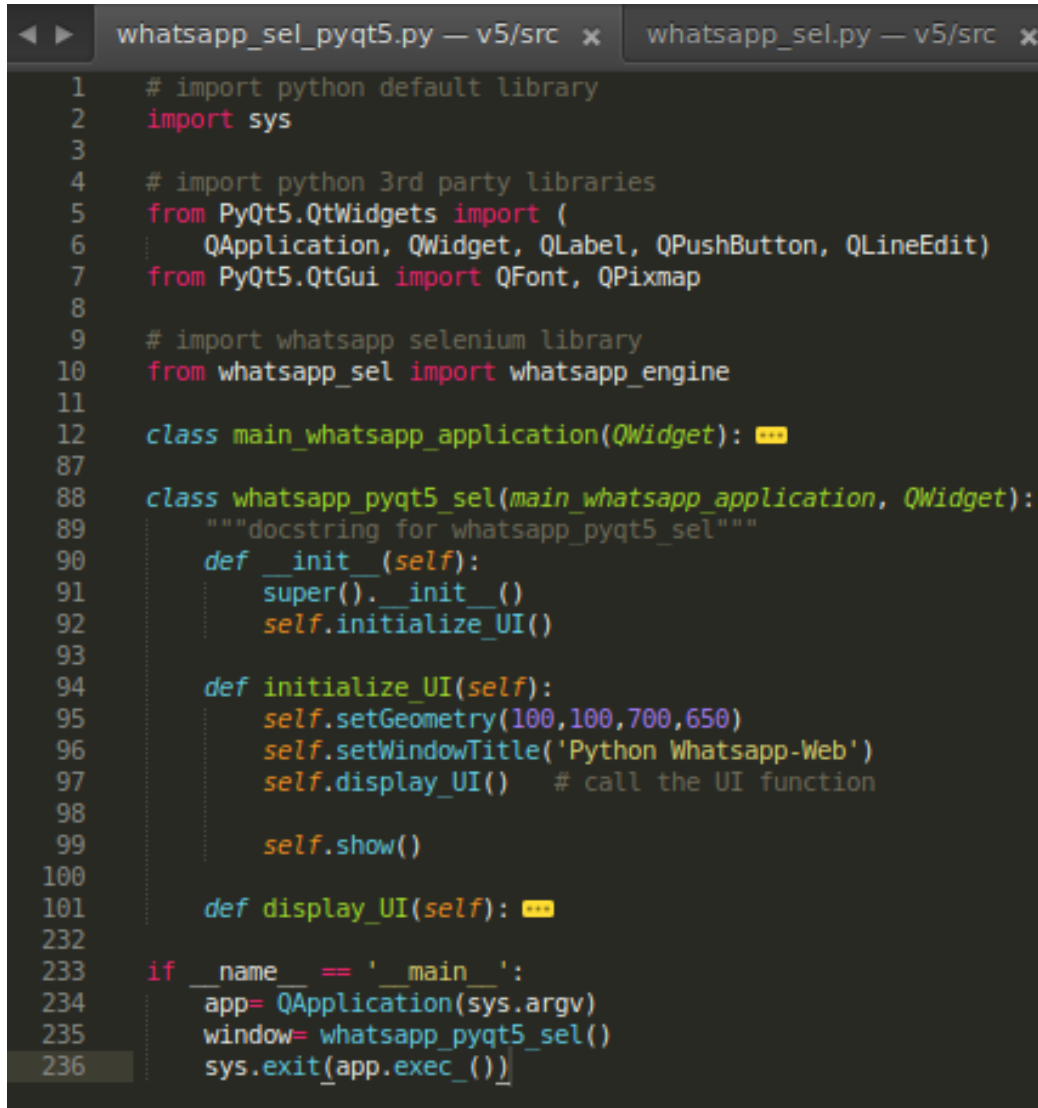


Figure 1: Python WhatsApp Web App GUI

1. User can click the the button and activate a chrome webdriver to start the WhatsApp Web page. The application will wait for the user to sign in by scanning the QR code in the WhatsApp web start page.
2. The second section is to provide the user to send message through the application platform. If no conversation yet been made previously, the app will launch a new conversation automatically.
3. The third section of the GUI receives a pool of contact numbers, a message from user, and send the message all at once to the provided numbers.
4. A button for the user to start the automation process. A contact number must be provided before the automation to start to let the program knows where to listen from.
5. The button at the bottom of the GUI is to end and close the browser properly with no any background process running behind.

The following in Figure 2 shows the PyQt5 code structure that is used to implement all those features and integrate everything in the application.



```
1  # import python default library
2  import sys
3
4  # import python 3rd party libraries
5  from PyQt5.QtWidgets import (
6      QApplication, QWidget, QLabel, QPushButton, QLineEdit)
7  from PyQt5.QtGui import QFont, QPixmap
8
9  # import whatsapp selenium library
10 from whatsapp_sel import whatsapp_engine
11
12 class main_whatsapp_application(QWidget): ...
13
14 class whatsapp_pyqt5_sel(main_whatsapp_application, QWidget):
15     """docstring for whatsapp_pyqt5_sel"""
16     def __init__(self):
17         super().__init__()
18         self.initialize_UI()
19
20     def initialize_UI(self):
21         self.setGeometry(100,100,700,650)
22         self.setWindowTitle('Python Whatsapp-Web')
23         self.display_UI() # call the UI function
24
25         self.show()
26
27     def display_UI(self): ...
28
29 if __name__ == '__main__':
30     app= QApplication(sys.argv)
31     window= whatsapp_pyqt5_sel()
32     sys.exit(app.exec_())
```

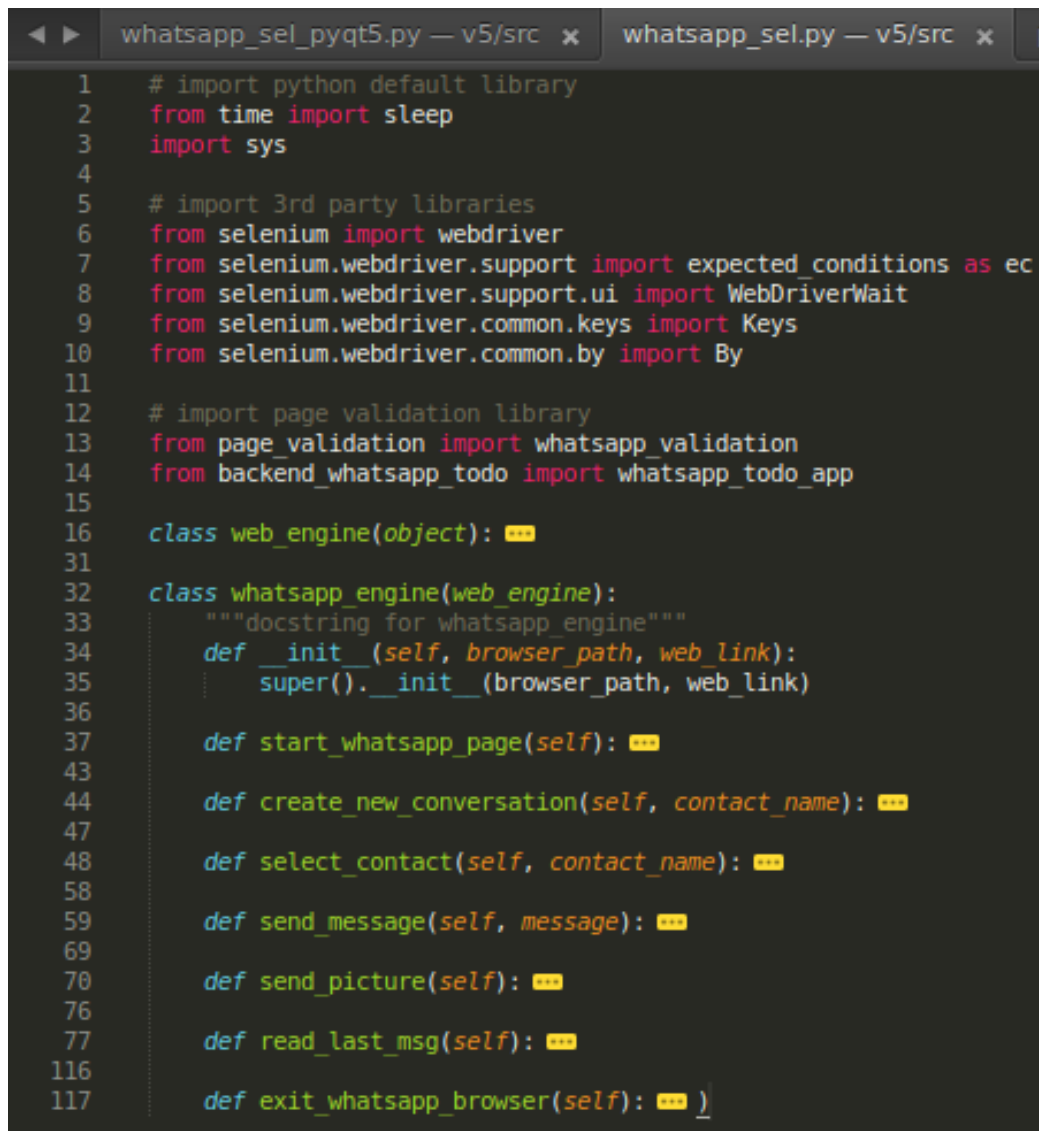
Figure 2: Implementation of PyQt5 code structure

3.2 Goal 2: Interconnecting Python with WhatsApp

Selenium webdriver objects are being utilized to sit between the Python script and the browser. This exposes the WhatsApp Web functionality to be used thus providing various automation processes that can be performed.

The application uses WhatsApp Web operations such as sending texts, selecting contact, and read messages so that various automation process can be achieved.

Figure 3 illustrates the implementation of Python selenium library to merge between Python script and WhatsApp Web.



```
1  # import python default library
2  from time import sleep
3  import sys
4
5  # import 3rd party libraries
6  from selenium import webdriver
7  from selenium.webdriver.support import expected_conditions as ec
8  from selenium.webdriver.support.ui import WebDriverWait
9  from selenium.webdriver.common.keys import Keys
10 from selenium.webdriver.common.by import By
11
12 # import page validation library
13 from page_validation import whatsapp_validation
14 from backend_whatsapp_todo import whatsapp_todo_app
15
16 class web_engine(object): ...
31
32 class whatsapp_engine(web_engine):
33     """docstring for whatsapp_engine"""
34     def __init__(self, browser_path, web_link):
35         super().__init__(browser_path, web_link)
36
37     def start_whatsapp_page(self): ...
43
44     def create_new_conversation(self, contact_name): ...
47
48     def select_contact(self, contact_name): ...
58
59     def send_message(self, message): ...
69
70     def send_picture(self): ...
76
77     def read_last_msg(self): ...
116
117     def exit_whatsapp_browser(self): ... )
```

Figure 3: Utilizing Python selenium library

3.3 Goal 3 & 4: WhatsApp Automation

From the graphical user interface (GUI), the user tells the Python script from which contact it should scan from. The Python script then will read the last message from the given contact and translates the user input into an operation based on what it has been set to.

Below in Figure 4 to Figure 6 show how the translating procedures being made.

```
63     def todo_app_call(self):
64         # calling todo app
65         try:
66             contact_name= self.contact_label_TODO_entry.text()
67             self.whatsapp_page.select_contact(contact_name)
68             # Fri 19 Feb 2021 08:24:12 PM AWST
69             while True:
70                 if self.whatsapp_page.read_last_msg() == None:
71                     print('Error occured reading chat timestamp.\nAutomation stopped.')
72                     break
73                 elif type(self.whatsapp_page.read_last_msg()) == list(): ...
74                 else:
75                     pass
76             except Exception as e:
77                 print('Error occured selecting contact. Automation stopped.\n', e)]
78
```

Figure 4: Calling WhatsApp to-do app method

```

77     def read_last_msg(self):
78         # xpath
79         read_target= '//span[@class="_1VzZY selectable-text copyable-text"]'
80         read_target= self.browser.find_elements_by_xpath(read_target)
81
82         # Fri 19 Feb 2021 11:44:39 AM AWST
83         last_messages= read_target[len(read_target)-1]
84         last_messages= last_messages.text
85
86         try:
87             timestamp_target= '//span[@class="_2JNr-"][@dir="auto"]'
88             timestamp_target= self.browser.find_elements_by_xpath(timestamp_target)
89             timestamp_value= timestamp_target[len(timestamp_target)-1]
90             timestamp_value= timestamp_value.text
91             print(timestamp_value)
92         except Exception as e:
93             print('Error reading timestamp',e)
94             return
95
96         # compare timestamp
97         if timestamp_value == self.previous_timestamp:
98             # pause 30 seconds
99             sleep(30)
100         else:
101             # update the new timestamp
102             self.previous_timestamp= timestamp_value
103             # call todo app
104             self.TODO_app= whatsapp_todo_app()
105             if type(self.TODO_app.start_todo(last_messages)) == list():
106                 return self.TODO_app.start_todo(last_messages)
107             else:
108                 pass
109
110         return True
111

```

Figure 5: Calling back-end to-do app

```

26     def start_todo(self, user_input):
27         self.todo_data()
28         user_commands= list(user_input.split(':'))
29         if user_commands[0] == 'todo-add':
30             self.add_task(user_commands[1])
31             print('New task added')
32         elif user_commands[0] == 'todo-list':
33             tasks_list= self.get_task()
34             tasks_list= [task.replace('\n', '') for task in tasks_list]
35             for task in tasks_list:
36                 print(task)
37         elif user_commands[0] == 'todo-delete':
38             self.delete_task()
39             print('Popleft a task!')
40         else:
41             print('No action taken.')

```

Figure 6: To-do app filtering user input

3.4 Goal 5: A Todo App as Scripting Example

A simple to-do app is carried out as an example of what can be done with this project. A defined set of commands have been put together for the user to perform with the application.

The tasks list are being saved by the to-do app into a text file saved in the system. User can just send text messages to the selected number and the Python script constantly reads every 30 seconds to scan for new automation job.

The to-do app consist off three operations that user can utilizes. The user defined operations are "todo-add" to add a new task and "todo-delete" to delete the first task in the list. Figure 7 illustrates the to-do app operations.

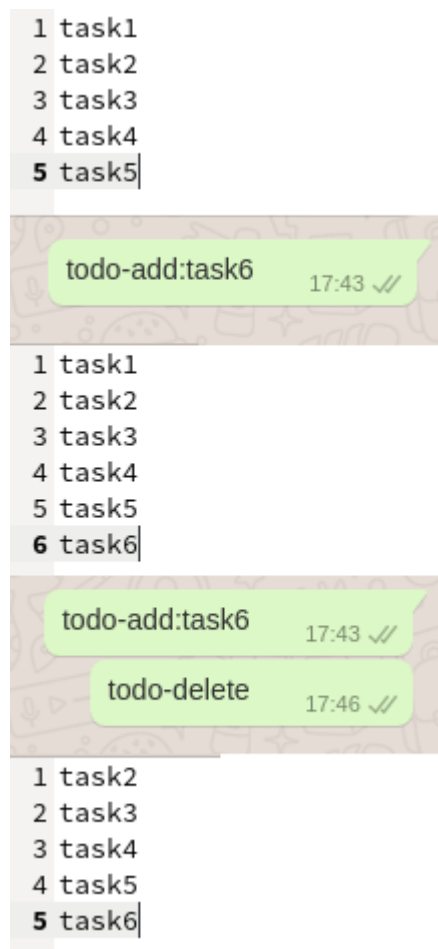


Figure 7: User defined commands from the to-do app

4 Discussion

4.1 What about other automation software?

The main take here is that it is not about automation per say, instead more on the user friendliness that is being put into effect. Non technical user can just sends a text message and Python script reads and translates the message into user command. The automation job activation process also very fast and trivial to use as it just consists of a simple text message through WhatsApp.

The to-do app that was being shown here is just an example of what can be done with the application. More examples are discussed below in section 4.3.

4.2 Why WhatsApp?

Why not? Just kidding! The reason for the WhatsApp Web implementation in the first place is just due to a project I previously worked on. Any other messaging app such as Telegram can also be put to use, just requires different approach. The rest is just the same. Pretty cool right!

4.3 Potentials

As I was saying in section 4.1 previously, the to-do app is just a gateway to plethora of automation that can be implemented.

Below are some examples of what can be done with the application:

Basic automation scripts:

1. Display text messages on a display board remotely.
2. Update customer details.
3. Update ticketing system.
4. Digital housekeeping automation.

Advanced automation scripts:

1. Company chatbot.
2. A terminal shell for the system.
3. Automate software testing.
4. WhatsApp as an organization ticketing system.

4.4 Security

Of course great power comes with great responsibility. As the application is intended for easy and user friendly, the Python script should not do any of automation that could jeopardize the system security.

As for more technical user, it is a powerful tool to have in hand. It is as useful as having a browser in a text message. Imagine sending a simple JSON text file through WhatsApp!

The limit of the implementation is just how secure you want the system need to be.

5 Appendix

A App Dependencies

Environment dependencies:

1. Python 3.x
2. Chrome browser installed in the system.

Additional Python modules (installation via pip install):

1. selenium
2. PyQt5
3. phonenumbers

B Improvements

Below are some of improvements that I currently have in mind:

1. A more complex graphical user interface with more features such as sending a picture.
2. Provide security measures by authenticating users.
3. More robust message reading technique so that the application can detects what operation not yet been done from a list of messages.