COMP411-Computer Network
Final Project: Automated email client and email log database
Jiaxuan Chen
Nov. 17th 2022

Introduction and Overview

As a program coordinator for the Office of International Student Affairs, I often find myself working on tasks like sending emails to over 100 recipients. While it is nearly impossible to send personal messages to every one of them, sending just a group email seems perfunctory. That is why I have I can create an auto email sender which provides me maximum flexibility in parsing the email content and ultimately schedules/ sends regular check-in emails to participants of the program automatically. My supervisor also requires me to log my communications with program participants for future optimization purposes, so I created a MySQL database in the hypothetical server in Linux virtual machine and it will grab the data from the email-sending script and store it in the MySQL table for future use.

Packets and Functionality for sending the email (auto_email_module.py)

In class, we went over the rough version of email sending in the terminal with several restrictions:

1. We cannot do cc or bcc to specify how we want the receivers to get the emails.
2. There are a lot of steps to follow to authenticate.
3. It is very hard to send attachments to enrich the message
4. You cannot actually send a scheduled email

According to these perspectives, I choose to import email and smtplib to set up a connection with the email server and parse the email with proper headers like sender, receivers, and attachments.

First, I used smtplib. SMTP to communicate with the email server. Since Google disabled access through a third-party connection. I choose the outlook server (“smtp.office365.com ”). Then I parse the email with the proper lines using email.MIMEmultipart allows adding headers like subject, from, cc, bcc, and to. I then attached the header lines to the body text. After that, I used email.MIMEApplication to grab the attachment and attach it properly to the message I just created. The helper _format_address is to format the email address in the correct encoded format so that it can be fed into the send_email function. I create a simple error code feature with a bunch of try-except blocks to make sure the user knows which step is off. The date_difference function works as a calculator and scheduler to make sure the email is sent at the scheduled time instead of the current time.

Error code:

01 Account authentication failure
02 Attachment Setting failure

MySQL database set up and connect (Navicat Premium, mysql_connect.py, init_email_log_autosend.sql)

First, I set up MySQL on the Linux virtual machine and try the "mysql -u root -p" to validate the local login.

For the simplicity of this project, I bypassed the authentication table in the built-in MySQL admin database by adding a comment in the mysql.config.d configuration file of MySQL, using skip-grant tables. To make sure of a smooth remote connection, I executed the command "GRANT ALL PRIVILEGES ON *.* TO 'root' @ '%' IDENTIFIED BY 'ADMIN_PASSWORD';
FLUSH PRIVILEGES; to grant root access to any user trying to connect. I also commented on the binding address so MySQL can listen to any host instead of just the host (binding_address=0.0.0.0).

Then I create the autosend_email_log database along with a table to store the log under that database. Finally, I use Pymysql to create a connection with the MySQL server on Linux so that it can write the email log automatically once the email is sent out.

Navicat Premium (https://www.navicat.com/en/) is a database admin allowing you to manually manage the database via the visual interface. It helps to validate if the log is being logged into the right table when I run the python script.

Finally, the email_log_autosend.sql is intended to initialize the table that stores all the logs.

Console file (auto_send_email.py)

In the auto_send_email.py, I manipulated the flags for different modules like scheduled send and undo timer which will be fed into the AutoEmail module. By calling send_email in the auto_email_module.py, we will send the email, and by calling update_mysql_email_log in mysql_connect.py, we can update the email logs in the MySQL database.

Notice that there will be two authentications, (module getpass is used to provide privacy when entering the password) the first one will be logging in to the email account that sends the email (in our case the outlook email) and the second one will be logging in to the MySQL server that stores the email.

Future work (init_cron_job_lst.py)

I believe one useful(advanced) feature is auto-scheduling the email to the clients (like sending birthday emails to participants). Ideally, I can finish the script init_cron_job_lst.py which puts the job into the cron-job list.