

SPAMMY

1. Introduction:

Spammy is an Email-Spammer interface which can be implemented in various applications to spam/send multiple emails to one or many users at the same time. The user can spam emails using their own account via our secure authentication feature. You can also send emails through a randomly generated account to maintain your privacy and anonymity. The email supports font-styling including bold/italics/underline etc. It also stores all the previously sent emails of an account in a database and allows the user to resend a previously sent email.

2. Literature Survey

We found this ever-existing ephemeral problem that organizations like colleges, companies, government offices etc, have to send mass emails to a wide range of audiences including their students, employees, etc.

Hence to solve this issue, we developed Spammy, an application which can be deployed on a web server to minimize the effort required to send these emails in bulk. The problem we found with normal email spammers which are already out there is that they do not use safe methods to connect to the email services. A few of the email-spammers we found on the internet, didn't even use SSL to ensure the safety of the SMTP messages.

Spammy is an Email-Spammer that uses the SSL protocol to ensure safety of the SMTP messages and is compatible with the GMAIL SMTP service. Thus, a user can effectively login to their Gmail accounts and send mass emails just by typing them out once.

Gmail has a feature to send mass emails, however you can send them only once. But using Spammy you can send these mass emails multiple times to any number of recipients.

Spammy also allows the users in a very user-friendly way, to save edit, save and delete their previous messages in order to reduce the effort and increase productivity.

3. Methodology and Implementation:

- The backend of the website is designed using the flask micro-web framework which is based off on python. As our application is a lightweight application, flask seemed like the best choice because it's fast as it has no database abstraction, form validation or any other third party components.

```
index.py — D:\Ap-lab-miniproject — Atom
File Edit View Selection Find Packages Help
index.py
1 from flask import *
2 from flask_session import Session
3 import sqlite3 as sql
4 import uuid
5 import smtplib, ssl, sys
6 import spammer_util as spamy
7
8 app = Flask(__name__)
9 conn = None
10 app.config["SESSION_PERMANENT"] = False
11 app.config["SESSION_TYPE"] = "filesystem"
12 Session(app)
13
14 class Email:
15     def __init__(self, email, uid, subject, nums, tlist, body):
16         self.email = email
17         self.uid = uid
18         self.sub = subject
19         self.nums = nums
20         self.tlist = tlist
21         self.body = body
22
23 @app.route("/", methods = ["GET", "POST"])
24 def index_page():
25     if session.get("email"):
26         return redirect('/home')
27     elif request.method == "POST":
28
D:\Ap-lab-miniproject\index.py 1:1 CRLF UTF-8 Python main Fetch GitHub Git (42) 1 update
```

(Example of our code with flask)

We have also used FLASK_SESSIONS library to create a session which stores the user's information for as long as the user is active

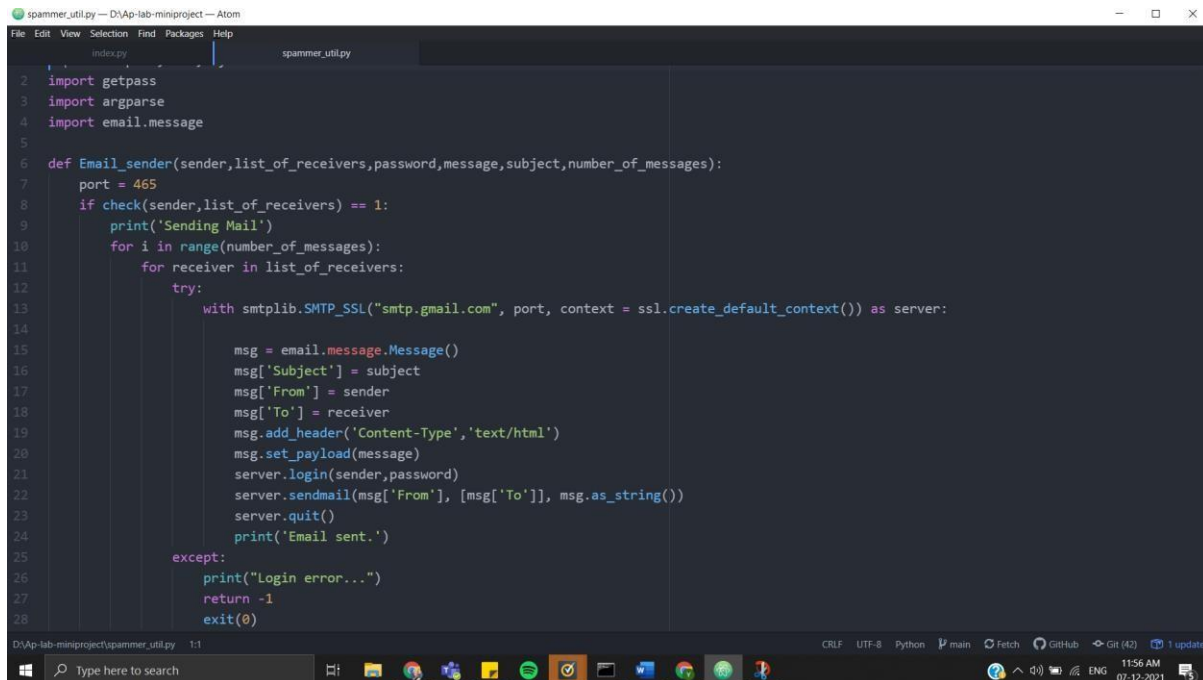
- We have used sqlite3 module for database support in our application. There is only one table with (UUID, Email_id, Subject , Number of emails, Body , target_list)

```
176 if __name__ == "__main__":
177     db = sql.connect('emails.db')
178     conn = db.execute(''' CREATE TABLE IF NOT EXISTS EMAIL(
179         EMAIL TEXT NOT NULL,
180         UID TEXT PRIMARY KEY,
181         SUBJECT TEXT NOT NULL,
182         NUMS INT NOT NULL,
183         TARGETS TEXT NOT NULL,
184         BODY TEXT NOT NULL); ''')
185     db.commit()
186     app.run(debug = True)
```

- For the front-end we have used html, css, js to design the templates which are rendered by the flask backend.

```
index.py spammer_util.py logout.css login.css
1 @import url('https://fonts.googleapis.com/css?family=Montserrat:400,800');
2 * {
3     box-sizing: border-box;
4 }
5
6 body {
7     background: #f6f5f7;
8     display: flex;
9     justify-content: center;
10    align-items: center;
11    flex-direction: column;
12    font-family: 'Montserrat', sans-serif;
13    height: 100vh;
14    margin: -20px 0 50px;
15 }
16
17 h1 {
18     font-weight: bold;
19     margin: 0;
20     font-size: 30px;
21 }
22
```

- For the main functionality to send the emails, we have used the SMTP module. This module is used to login to the Gmail service and then send the email. We also use the email.message module to support HTML message rendering.



```
1 spammer_util.py — D:\Ap-lab-miniproject — Atom
2 import getpass
3 import argparse
4 import email.message
5
6 def Email_sender(sender,list_of_receivers,password,message,subject,number_of_messages):
7     port = 465
8     if check(sender,list_of_receivers) == 1:
9         print('Sending Mail')
10        for i in range(number_of_messages):
11            for receiver in list_of_receivers:
12                try:
13                    with smtplib.SMTP_SSL("smtp.gmail.com", port, context = ssl.create_default_context()) as server:
14
15                        msg = email.message.Message()
16                        msg['Subject'] = subject
17                        msg['From'] = sender
18                        msg['To'] = receiver
19                        msg.add_header('Content-Type','text/html')
20                        msg.set_payload(message)
21                        server.login(sender,password)
22                        server.sendmail(msg['From'], [msg['To']], msg.as_string())
23                        server.quit()
24                        print('Email sent.')
25                except:
26                    print("Login error...")
27                    return -1
28            exit(0)
```

Method:

The user can enter his details or use a random account to spam emails.

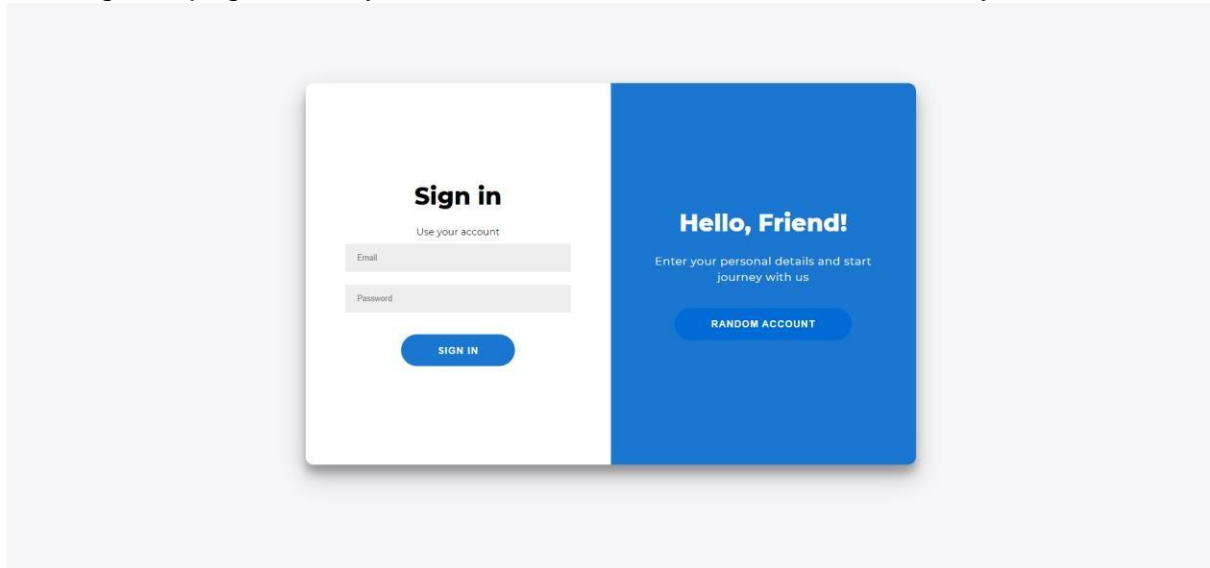
If he enters his details, we will create a session and store his email. Then we will allow him to enter the new mail and enter the number of times to spam it. We also give the user an option to re-use previous emails from our database.

If the user decides to use a random account to spam messages, then he will directly be redirected to the email editing page and he can enter a new mail and spam number

When the user presses the SPAM button, the email sending function in the backend is triggered and the email is sent to the specified recipients.

4. Results

The Sign- In page where you can choose a random account or enter your details:



The Create Mail page:

Spammy Manage Emails Create New Email





Create Mail

Email/Emails

Subject

Body

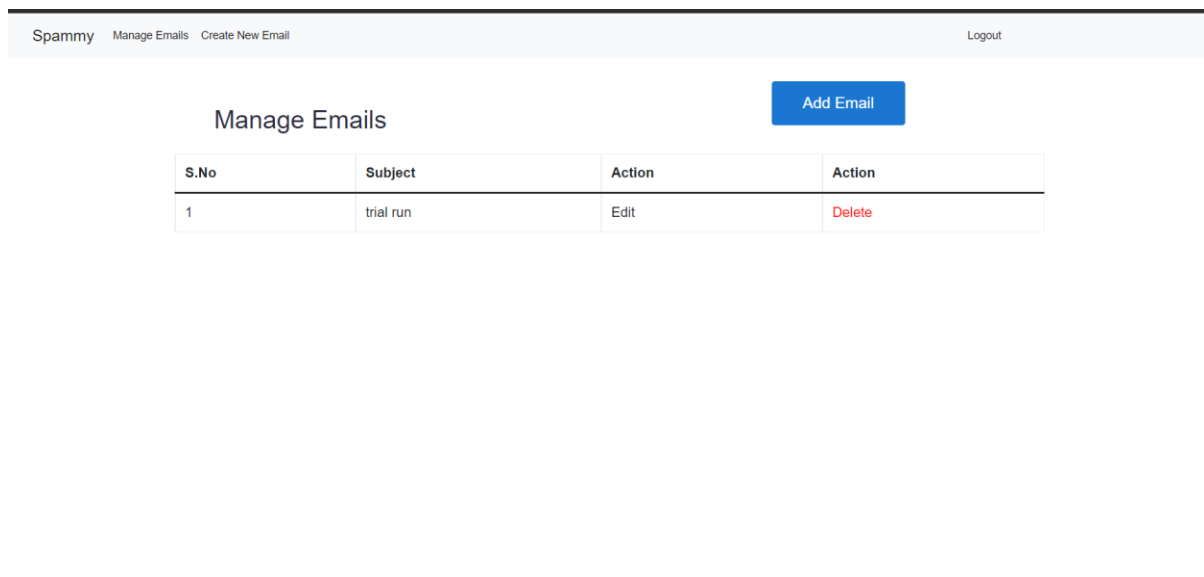
Paragraph

B *I*    

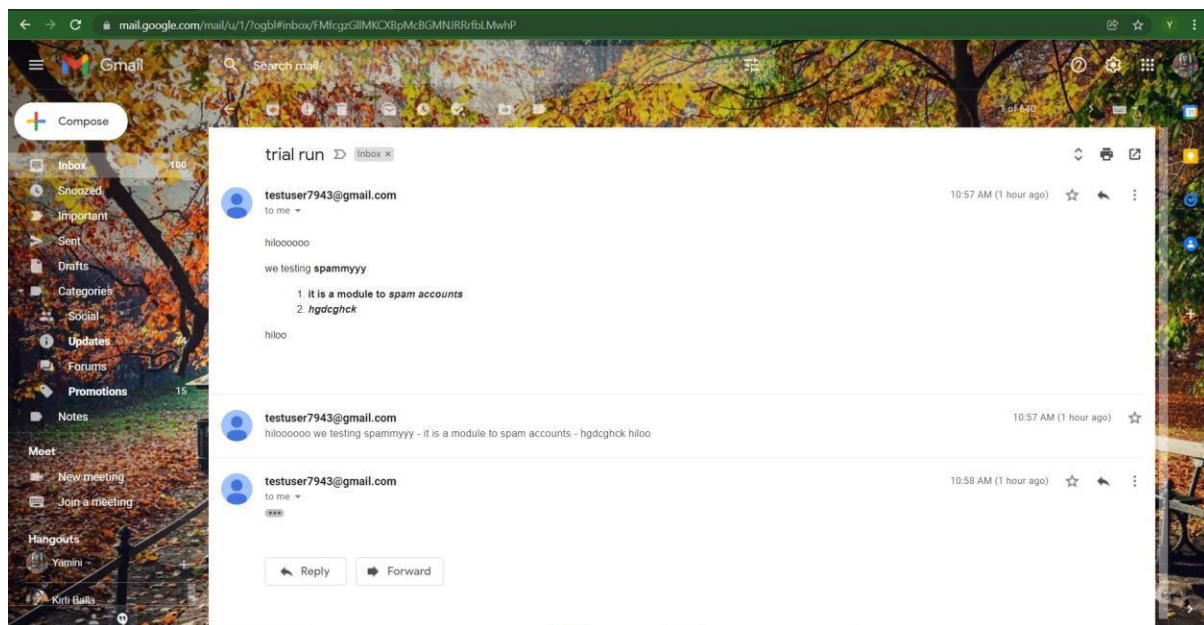
number of emails

Spam

The Manage Emails page:



The Final results on the Gmail account which has been spammed:



5. Conclusion

Thus, we can use this application for sending mass emails quickly and efficiently and can be very beneficial in many organizational use cases. It is secure, has a seamless user experience and provides you with much needed anonymity while saving your precious time.