Automated E-mail Sender

Introduction -:

Certainly, an automated email sender is a script or program that sends emails automatically without the need for manual intervention. It's a useful tool for various purposes, such as sending notifications, reports, newsletters, or any other recurring email tasks. Here are some key aspects and use cases of automated email senders:

Key Aspects -:

- **Scripting or Programming**: Automated email senders are typically created using a programming language like Python, Java, C#, etc. These scripts interact with email servers using protocols like SMTP (Simple Mail Transfer Protocol).

- **Email libraries: -** Programming libraries like smtplib (for Python) or built-in functions for handling emails are often used to compose, format, and send emails programmatically.

- **Configuration**: - You need to configure the email sender with your email credentials, including your email address and password (or API keys for services like SendGrid).

- **Recipient List: -** Automated email senders can send emails to one or multiple recipients. The recipient list can be hard-coded or loaded dynamically from a file or database.

- **Subject and Content: -** You can customize the subject, body, and attachments of the email based on your requirements.

- **Scheduling:** You can schedule automated emails to be sent at specific times or on recurring schedules, like daily, weekly, or monthly.

Use Cases-:

- **Notifications-:** Automated email senders are commonly used to send notifications for events such as server downtime, errors in applications, or updates on completed tasks.

- **Reports-:** Businesses often use automated email senders to generate and send reports, such as sales reports, financial summaries, or analytics reports to stakeholders.

- **Marketing-:** Email marketing campaigns can be automated to send newsletters, promotional offers, and updates to a list of subscribers.

- **Reminders-:** Personalized reminders for appointments, birthdays, or other important dates can be sent automatically.

- **Transaction Emails-:** E-commerce platforms use automated emails to confirm orders, provide shipping information, and request reviews.

- **Feedback Requests-:** After a user interaction or purchase, automated emails can be sent to gather feedback or reviews.

- **Password Resets-:** Automated password reset emails are sent when a user requests to reset their password.

- **Subscription Renewals-:** Subscription-based services can use automated emails to remind users of upcoming renewals or expirations.

Best Practices-:

- ✓ **Security-:** Ensure the security of your email credentials, and consider using environment variables or configuration files to store sensitive information.

- ✓ **Opt-Out Mechanism-:** For marketing emails, include an opt-out mechanism (unsubscribe link) to comply with anti-spam regulations like CAN-SPAM Act (in the U.S.) or GDPR (in the EU).

- ✓ **Testing-:** Thoroughly test your automated email sender to avoid issues such as spam classification, broken links, or incorrect content.

- ✓ **Logging-:** Implement logging to keep track of sent emails, errors, and delivery status.

- ✓ **Compliance-:** Familiarize yourself with email regulations and ensure compliance with them.

Automated email senders can streamline communication and save time in various scenarios, but it's important to use them responsibly and securely to avoid potential issues and ensure that your emails are well-received by recipients.

Installation Requirements-:

Hardware Requirements-:

- Computer (any modern computer should suffice)

Software Requirements

- Operating System: Windows 10, macOS 10.15, or Ubuntu 18.04
- Python 3.8 or higher

Required Python Libraries: csv, Datetime, email, smtplib, tkinter, sys, email. mime. Text, email.mime.multipart, MySQL. Connector

- Ram 4 GB and above
- 64-bit Processor

Optional Requirements

- Email Account: Access to an email account (e.g., Gmail)

- Third-party Services: If you plan to use additional services, such as weather APIs, you may need to obtain API keys.

Installation Steps

Install Python:

- Download and install Python from [Python's official website](https://www.python.org/downloads/).
- Ensure you check the option to add Python to the system PATH during installation.

Install Required Python Libraries:

- Open a terminal or command prompt.
- Run the following command to install the necessary Python libraries:
- Bash
- pip install smtplib email

Prepare Your Email Account:

- Make sure you have access to the email account you want to use for sending emails.
- Note down your email address and password.

Download and Run the Automated Email Sender:

- Download the automated email sender software package from [GitHub](https://github.com/your-project/repository).
- Follow the software's user guide to configure and use it effectively.

Troubleshooting

- If you encounter any issues during installation or usage, refer to the troubleshooting section in the user guide or seek support from the project's maintainers.

Step By Step Documentation: -

Automated-Email-Sender

1) Import Required Modules:

Import necessary Python modules, including csv, datetime, email, smtplib, tkinter, sys, email.mime.text, email.mime.multipart, and MySQL. Connector.

1.csv (Comma-separated Values):

Purpose: The csv library is used to read CSV files containing recipient email addresses.

Usage: It provides functions for parsing and manipulating CSV data.

2. datetime:

Purpose: The datetime library is used to work with date and time information.

Usage: It is used to timestamp when emails are sent and track the sent timestamps in the MySQL database.

3. email:

Purpose: The email library is used for creating and formatting email messages.

Usage: It helps in creating MIME (Multipurpose Internet Mail Extensions) messages, which are required for sending emails with attachments and structured content.

4. smtplib (Simple Mail Transfer Protocol Library):

Purpose: The smtplib library is used for sending email messages using the SMTP protocol.

Usage: It allows the script to connect to an SMTP server, authenticate, and send email messages.

5. tkinter:

Purpose: The tkinter library is the standard GUI (Graphical User Interface) library for Python.

Usage: It is used to create the graphical user interface for the email sender application, providing elements such as windows, buttons, labels, and text boxes.

6. sys:

Purpose: The sys library provides access to some variables used or maintained by the interpreter and to functions that interact with the interpreter.

Usage: In this script, it's imported but not explicitly used.

7. email.mime.text:

Purpose: The email.mime.text module is part of the email library and is specifically used for creating plain text email content.

Usage: It allows the script to create the text content of the email messages.

8. email.mime.multipart:

Purpose: The email.mime.multipart module is part of the email library and is used for creating multipart email messages.

Usage: It is used to create email messages with multiple parts, such as plain text and HTML.

9. MySQL. Connector:

Purpose: The MySQL. Connector library is used to interact with MySQL databases.

Usage: It is used to establish a connection to a MySQL database, create a tracking table, and record email tracking data in the MySQL database.

2) SMTP Configuration:

Set the SMTP server configuration, including server address, port, SMTP username, SMTP password, and sender email address.

3) Initialize Recipient List and MySQL Connection:

Create an empty list called recipient list to store recipient email addresses.

Establish a connection to a MySQL database named "email" with no password.

4) Create a MySQL Table:

Define a function create_tracking_table() to create a table named email tracking in the MySQL database if it doesn't already exist.

This table will store tracking information for sent emails, including recipient email, sent timestamp, delivery status, open status, and click-through status.

5) Record Email Tracking Data:

Define a function record_email_tracking() to insert email tracking data into the email tracking table in the MySQL database.

6) Display Email Tracking:

Define a function display_email_tracking() to open a new window using Tkinter to display email tracking details from the email tracking table in the MySQL database.

7) GUI Initialization:

Create the main GUI window using Tkinter.

Set the window title and dimensions.

8) GUI Elements:

Create various GUI elements within the main window, including labels, buttons, entry fields, and a text box.

These elements are used for importing recipient emails from a CSV file, adding/removing recipients manually, specifying email subject and body, sending emails, and displaying email tracking information.

9) Import CSV Function:

Define the import_csv() function, which allows the user to select a CSV file containing recipient email addresses and import them into the recipient list.

10) Update Recipient List box:

Define the update_recipient_listbox() function to refresh the list of recipients displayed in the GUI list box.

11) Add Recipient Function:

Define the add_recipient() function to add a recipient email manually to the recipient list and update the list box.

12) Remove Recipient Function:

Define the remove_recipient() function to remove a selected recipient from the recipient list and update the list box.

13) Send Emails Function:

Define the send_emails() function to send emails to the recipients in the list using the configured SMTP server.

For each recipient, it records the sent timestamp and the delivery status (whether it was delivered successfully or not) in the MySQL database.

14) Main Loop:

Start the Tkinter main loop to run the GUI application.

15) Close MySQL Connection: Close the MySQL database connection when the main loop exits.

Source code-:

```
import csv
import datetime
import email
import smtplib
import tkinter as tk
import sys
from tkinter import RIDGE, Frame, filedialog, messagebox
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import mysql.connector
from datetime import datetime

# Set your SMTP configuration here
smtp_server = "smtp.gmail.com"
smtp_port = 587
smtp_username = "chetanvaishnav378@gmail.com"
smtp_password = "dozw bncy usto ngub"
sender_email = "chetanvaishnav378@gmail.com"

# Initialize an empty recipient list
recipient_list = []

mysql_connection = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="email"
)

# Function to create a tracking table if it doesn't exist
def create_tracking_table():
    cursor = mysql_connection.cursor()
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS email_tracking (
            id INT AUTO_INCREMENT PRIMARY KEY,
            recipient_email VARCHAR(255),
            sent_timestamp DATETIME,
            delivery_status VARCHAR(255),
            open_status BOOLEAN DEFAULT 0,
            click_through BOOLEAN DEFAULT 0
        )
    """)
    mysql_connection.commit()

# Function to record email tracking data in MySQL
```

```
def record_email_tracking(recipient_email, sent_timestamp,
delivery_status):
    cursor = mysql_connection.cursor()
    cursor.execute("""
        INSERT INTO email_tracking (recipient_email, sent_timestamp,
delivery_status)
        VALUES (%s, %s, %s)
    """, (recipient_email, sent_timestamp, delivery_status))
    mysql_connection.commit()

def display_email_tracking():
        try:
        # Open a new window to display the email tracking details
            tracking_window = tk.Toplevel()
            tracking_window.title("Email Tracking Details")
            tracking_window.geometry("800x600")

        # Create a Text widget to display tracking details
            tracking_text = tk.Text(tracking_window, wrap=tk.WORD,
height=20, width=60)
            tracking_text.pack(padx=20, pady=20)

        # Retrieve data from the email_tracking table
            cursor = mysql_connection.cursor()
            cursor.execute("SELECT recipient_email, sent_timestamp,
delivery_status FROM email_tracking")
            tracking_data = cursor.fetchall()

        # Populate the Text widget with tracking details
            for row in tracking_data:
                recipient_email, sent_timestamp, delivery_status = row
                tracking_text.insert(tk.END, f"Recipient:
{recipient_email}\n")
                tracking_text.insert(tk.END, f"Sent Timestamp:
{sent_timestamp}\n")
                tracking_text.insert(tk.END, f"Delivery Status:
{delivery_status}\n")
                tracking_text.insert(tk.END, "\n")

        except Exception as e:
            pass

# Function to browse for a CSV file and import recipients
def import_csv():
    file_path = filedialog.askopenfilename(filetypes=[("email", "*.csv")])
    if file_path:
        try:
            with open(file_path, "r") as csvfile:
```

```
                reader = csv.DictReader(csvfile)
                recipient_list.clear()  # Clear existing recipients
                for row in reader:
                    recipient_list.append(row)
                update_recipient_listbox()
        except:
            messagebox.showinfo("message","message : Error importing CSV")

# Function to update the recipient listbox
def update_recipient_listbox():
    recipient_listbox.delete(0, tk.END)
    for recipient in recipient_list:
        recipient_listbox.insert(tk.END, recipient["Email"])

# Function to add a recipient
def add_recipient():
    email = recipient_entry.get().strip()
    if email:
        recipient_list.append({"Email": email})
        update_recipient_listbox()
        recipient_entry.delete(0, tk.END)

# Function to remove a selected recipient
def remove_recipient():
    selected_index = recipient_listbox.curselection()
    if selected_index:
        recipient_list.pop(selected_index[0])
        update_recipient_listbox()

# Function to send emails
def send_emails():
    global mysql_connection
    if not recipient_list:
        messagebox.showinfo("message : unsuccefully sending email")
        return

    subject = subject_entry.get()
    email_body = body_text.get("1.0", tk.END)

    try:
        with smtplib.SMTP(smtp_server, smtp_port) as server:
            server.starttls()
            server.login(smtp_username, smtp_password)

            for recipient in recipient_list:
                recipient_email = recipient["Email"]
                message = MIMEMultipart()
                message["From"] = sender_email
```

```
                message["To"] = recipient_email
                message["Subject"] = subject

                email_text = MIMEText(email_body, "plain")
                message.attach(email_text)

                try:
                    # Record the timestamp when the email is sent
                    sent_timestamp = datetime.now()

                    server.sendmail(sender_email, recipient_email,
message.as_string())

                    # Record successful email delivery
                    record_email_tracking(recipient_email, sent_timestamp,
"Delivered")

                    print("Email sent successfully")

                except smtplib.SMTPException as e:
                    messagebox.showinfo("info","Done")
                    break
    except mysql.connector.Error as err:
        messagebox.showinfo("Error", f"MySQL Error: {str(err)}")
    except:
        messagebox.showinfo("Error", f"An error occurred: {str(email)}")
    finally:
        if mysql_connection:
            mysql_connection.close()


root = tk.Tk()
root.title("Email Sender") # type: ignore
root.configure(bg='light pink')
root.geometry("1350x1000+0+0")

# Create and place a title label
title_label = tk.Label(root, text="Automated-Email-Sender", font=("The
Times Of Roman", 28))
title_label.pack(pady=30)  # Add some padding at the top

frame1=Frame(root,bd='5',relief=RIDGE,bg='cyan')
frame1.pack(padx=20, pady=50)

# Create and place GUI widge
file_path_label = tk.Label(frame1, text="Recipient List (CSV):",font=("The
Times Of Roman", 10))
file_path_label.pack()
```

```
file_path_button = tk.Button(frame1, text="Import CSV", command=import_csv,
bg="magenta", font=("The Times Of Roman", 12))
file_path_button.pack()

recipient_label = tk.Label(frame1, text="Recipients:", font=("The Times Of
Roman", 10))
recipient_label.pack()

recipient_listbox = tk.Listbox(frame1, height=5, width=40,
selectmode=tk.SINGLE,bg="light yellow")
recipient_listbox.pack()

recipient_entry = tk.Entry(frame1)
recipient_entry = tk.Entry(frame1, bg="light yellow")
recipient_entry.pack()

add_button = tk.Button(frame1, text="Add Recipient", font=("The Times Of
Roman", 12), command=add_recipient, bg="magenta")
add_button.pack()

remove_button = tk.Button(frame1, text="Remove Recipient",
command=remove_recipient, bg="magenta", font=("The Times Of Roman", 12))
remove_button.pack()

subject_label = tk.Label(frame1, text="Subject:", font=("The Times Of
Roman", 10))
subject_label.pack()
subject_entry = tk.Entry(frame1)
subject_entry = tk.Entry(frame1, bg="light yellow")
subject_entry.pack()

body_label = tk.Label(frame1, text="Email Body:", font=("The Times Of
Roman", 10))
body_label.pack()
body_text = tk.Text(frame1, height=5, width=40, bg="light yellow")
body_text.pack()

send_button = tk.Button(frame1, text="Send Emails", command=send_emails,
bg="magenta", font=("The Times Of Roman", 12))
send_button.pack()

display_tracking_button = tk.Button(frame1, text="Display Email Tracking",
command=display_email_tracking, bg="red", font=("The Times Of Roman", 12))
display_tracking_button.pack()

create_tracking_table()

root.mainloop()
```

```
mysql_connection.close()
```

Output-: