Auto-Emailer Algorithm and Implementation Report

Your Name

March 7, 2025

Contents

1 Introduction

This report provides a detailed explanation of the Auto-Emailer algorithm and its implementation. The script is designed to send personalized emails to a list of recipients with customizable content and attachments. It supports multiple languages and uses multi-threading to manage email sending efficiently.

2 Algorithm Overview

The Auto-Emailer algorithm can be broken down into several key components:

- Reading and parsing contact information from a CSV file.
- Loading email templates for different languages.
- Formatting email content with recipient-specific information.
- Sending emails with optional attachments.
- Managing multiple email sending threads to improve efficiency.

3 Detailed Algorithm Explanation

3.1 Reading Contact Information

The algorithm starts by reading contact information from a CSV file. The CSV file is expected to contain at least the email addresses of the recipients. Optionally, it can include names and preferred languages.

3.2 Loading Email Templates

Email templates are loaded from text files. Each template corresponds to a different language. The algorithm selects the appropriate template based on the recipient's preferred language.

3.3 Formatting Email Content

The email content is formatted by replacing placeholders (e.g., [NAME]) with the recipient's specific information. This ensures that each email is personalized.

3.4 Sending Emails

The algorithm sends emails using the SMTP protocol. It handles attachments by encoding them in the email. Multi-threading is used to send multiple emails simultaneously, improving efficiency.

3.5 Error Handling and Retries

The algorithm includes mechanisms for handling errors and retrying failed email sends. This ensures robustness and reliability.

4 Implementation Details

4.1 Code Structure

The code is structured into several functions, each handling a specific part of the algorithm. Below is a detailed explanation of each function.

4.1.1 Reading CSV and Text Files

```
def read_csv(file_path):
    with open(file_path, mode='r', newline='', encoding='utf-8') as file:
        return list(csv.reader(file))

def read_txt(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        return file.read()
```

4.1.2 Sending Emails

```
def send_email(subject, body, recipient, attachments=[]):
    """Function to send an email to a single recipient."""
    for attempt in range(RETRIES + 1):
        try:
            msg = MIMEMultipart()
            msg['From'] = EMAIL_ADDRESS
            msg['To'] = recipient
            msg['Subject'] = subject

            # Attach email body with UTF-8 encoding
            msg.attach(MIMEText(body, 'plain', 'utf-8'))

            for attachment in attachments:
                try:
                    with open(attachment, 'rb') as file:
                        part = MIMEBase('application', 'octet-stream')
                        part.set_payload(file.read())
                        encoders.encode_base64(part)
                        part.add_header('Content-Disposition', f'attachment; filename={
    os.path.basename(attachment)}')
                        msg.attach(part)
                except Exception as e:
                    print(f'Failed to attach file {attachment}: {e}')

            with smtplib.SMTP('smtp.gmail.com', 587, timeout=TIMEOUT) as server:
                server.starttls()
                server.login(EMAIL_ADDRESS, EMAIL_PASSWORD)
                server.send_message(msg)
                print(f'Email sent to {recipient}')
            return

        except Exception as e:
            print(f'Failed to send email to {recipient} (Attempt {attempt+1}/{RETRIES
    +1}): {e}')
            if attempt < RETRIES:
                time.sleep(PAUSE_BETWEEN_ATTEMPTS)
```

5 Performance Considerations

The algorithm is designed to be efficient, with multi-threading allowing for parallel email sending. However, performance can be affected by network conditions and the SMTP server's limitations.

6 Conclusion

The Auto-Emailer algorithm is a robust and efficient solution for sending personalized emails to a list of recipients. Its modular design and error-handling mechanisms ensure reliability and ease of use.