AUTOMATED SOCIAL MEDIA & EMAIL POSTING USING FLASK AND API INTEGRATION

A project work submitted in the partial fulfilment of the requirement for the award of the degree

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

SK. MD. HUSSAIN	**L22CSE279011**	**O. SRINIVASA RAO**	**L22CSE279008**
M. BAIDHYUTH	**Y21CSE279068**	**N. VAMSI KESAVA**	**Y21CSE279082**

Under the Esteemed Guidance of

Dr. M. RAGHAVA NAIDU

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KRISHNA UNIVERSITY COLLEGE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, New Delhi)

MACHILIPATNAM-521 004

2024-2025

KRISHNA UNIVERSITY

COLLEGE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, NEW DELHI)

MACHILIPATNAM-521004

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled "**Automated Social media & Email Posting Using Flask and API Integration**" is a Bonafide work done by

SK. MD. HUSSAIN	**L22CSE279011**	**O. SRINIVASA RAO**	**L22CSE279008**
M. BAIDHYUTH	**Y21CSE279068**	**N. VAMSI KESAVA**	**Y21CSE279082**

Submitted in partial fulfilment of the requirements for the award of the degree of **"BACHELOR OF TECHNOLOGY IN COMPUTER SCIENC& ENGINEERING"** during the academic year **2024-2025.**

Project Guide **Head of The Department**

External Examiner

DECLARATION

This is to certify that the work reported in this titled **"Automated Social media and Multi Email Posting Using Flask and API Integration"**, was submitted in the Department of **COMPUTER SCIENCE & ENGINEERING(CSE**). Krishna University College of Engineering and Technology, Machilipatnam in the partial fulfilment of degree for the award of Bachelor of Technology, and is a Bonafide work done by us the reported results are based on the project week entirely done by us and not copied on any other source also, we declare that the matter embedded in this thesis has not been submitted by us in full or partial thereof for the award of any degree/diploma of any other institution or University previously.

Yours Sincerely,

SK.MD. HUSSAIN	**L22CSE279011**
O. SRINIVASA RAO	**L22CSE279008**
M. BAIDHYUTH	**Y21CSE279068**
N. VAMSI KESAVA	**Y21CSE279082**

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts with success. This acknowledgement transcends the reality of formality when we would like to express deep gratitude and respect to all those people behind the screen who guided, inspired and helped us for the completion of my project work.

We are extremely grateful to our guide **Dr. M. Raghava Naidu, Department of Computer Science & Engineering, Krishna University College of Engineering and Technology,** for giving us an opportunity to work on a project with his timely suggestions and guidance.

We express our sincere gratitude to **Mr. P. Bhavani Shankar, Assistant Professor and Project Coordinator, Department of Computer Science & Engineering, Krishna University College of Engineering and Technology**, for this valuable guidance, motivation, and support throughout this period

We are extremely grateful to **Mr. M. Sai Somayajulu, Assistant Professor and Coordinator of Computer Science & Engineering Department, Krishna University College of Engineering and Technology,** for his motivation and valuable advice during this period.

We are extremely grateful to **Dr. R. Vijaya kumari, Principle(i/c) and Head of the Department, Computer Science & Engineering, Krishna University College of Engineering and Technology,** for her motivation and valuable advice during this period.

We express our profound thanks to **Prof. Ramji Koona, Vice Chancellor, Prof. M. V. Basaveswara Rao, Rector**, **Prof. N. Usha, Registrar, Krishna University and Prof. Y. K. Sundara Krishna, Dean, Faculty of Engineering and Technology** for their valuable motivation and support in completing our project work.

We express our heartful thanks to all our Teaching and Non-Teaching staff.

Sincerely,

SK. MD. HUSSAIN	**L22CSE279011**
O. SRINIVASA RAO	**L22CSE279008**
M. BAIDHYUTH	**Y21CSE279068**
N. VAMSI KESAVA	**Y21CSE279082**

ABSTRACT

The “**Automated Social media and Multi Email Posting Using Flask and API Integration**”. In today’s fast-paced digital world, timely and scalable communication is crucial for both individuals and organizations. This project introduces a unified, **web-based platform** that automates the process of sending **bulk emails** and posting content to social media—**built using Flask and integrated** with various **API** s. The platform allows users to log in securely using email credentials or social media access tokens. Once authenticated, users can upload recipient lists and media files, write personalized messages, and send them out efficiently. Whether it’s emails with attachments or social media posts with images and videos, the system ensures smooth and streamlined communication. The backend is powered by **Flask**, while the frontend uses **HTML, CSS**, and **JavaScript** for a responsive and user-friendly experience. Real-time feedback, delivery tracking, and error handling are built in, reducing the chances of manual errors and saving time. Social media integration supports platforms like Facebook, Instagram, and LinkedIn—allowing posts to be scheduled or published instantly using secure credentials such as **page IDs or URN** s. By automating routine communication tasks, this solution boosts productivity, ensures consistency, and gives users full control over their outreach. It’s a powerful tool for marketing teams, organizational announcements, and professional networking—offering a secure, scalable, and cost-effective alternative to relying on third-party platforms.

Key Words:

Automated Communication, Bulk Emailing, Social media Posting, Flask, API Integration, Email Automation, Content Scheduling, Real-time Feedback, Secure Authentication, Facebook API, Instagram API, LinkedIn API, Recipient List Upload, Media File Upload, Message Personalization, Delivery Tracking, Scalable Communication, Marketing Automation, Web-based Platform, Cost-effective Solution

INDEX

INTRODUCTION

INTRODUCTION

The **Automated Multi-Email Posting System** addresses the repetitive and time-consuming nature of manual email communication by providing a streamlined, web-based platform for sending bulk emails efficiently and securely. This system enables users to log in with their email and API credentials, select recipient lists from uploaded .txt files, compose custom messages, and attach relevant files—all from a single, intuitive interface.

Designed for individuals and organizations that require regular email outreach, the system automates the email dispatch process and tracks successful transmissions, significantly reducing human effort and the risk of error. A standout feature of the platform is its ability to dynamically integrate user-specific credentials, eliminating the dependency on third-party services and ensuring secure, flexible email posting.

Developed using **Flask** for the backend and **HTML, CSS, and JavaScript** for the responsive frontend, the system delivers a seamless and interactive user experience. It empowers users to focus on the content and strategy of their communication, while the platform efficiently manages distribution. This makes the solution ideal for marketing campaigns, notifications, organizational updates, and any context requiring scalable email communication.

1. Problem Statement

Traditional methods of sending bulk emails are often manual, time-consuming, and prone to errors. This includes issues like missed recipients, incorrect attachments, and inconsistent delivery timing. These approaches lack automation, personalization, and scalability—limiting their effectiveness for users who need to reliably and quickly deliver information to large audiences.

2. Background

As email remains a primary channel of communication in both professional and organizational settings, the demand for automated solutions is increasing. Manual email management becomes inefficient and unsustainable at scale. Automated email systems help streamline the communication process by allowing users to upload recipient lists, compose personalized messages, and dispatch emails in bulk from a centralized interface—saving time, reducing errors, and improving overall productivity.

3. Motivation of the Project

The motivation behind this project is to create a robust, scalable solution that supports reliable bulk email communication without relying on costly third-party platforms. Professionals and organizations often face challenges in maintaining timely, accurate communication with large groups. This system aims to address those challenges by providing a secure, user-friendly interface that automates the bulk email process and ensures consistent delivery, accuracy, and ease of use.

4. Objectives

- To develop a web-based platform for automated bulk email sending.
- To enable secure login with user-specific email credentials and APIs.
- To support uploading of recipient lists via .txt files.
- To allow intuitive message composition with support for file attachments.
- To provide real-time feedback during the email dispatch process.
- To reduce manual errors and streamline communication workflows.

SYSTEM REQUIREMENTS

SYSTEM REQUIREMENTS

1. Hardware Requirements:

- **Processor:** Intel Core i3 or above
- **RAM:** 4 GB or higher
- **Hard Disk:** Minimum 100 MB free space
- **Display:** 1024x768 resolution or higher

2. Software Requirements:

- **Backend:** Python (Flask)
- **Frontend:** HTML, CSS, JavaScript
- **Database:** SQLite
- **Email Handling Libraries:** smtplib, email, ssl
- **Web Browser:** Google Chrome / Mozilla Firefox
- **Operating System:** Windows / Linux / macOS

SYSTEM ANALYSIS

SYSTEM ANALYSIS

3.1 Existing System

Current systems for bulk email messaging typically rely on manual methods or third-party services.

- **Email Systems:** Platforms such as Microsoft Outlook, Gmail, and Mailchimp allow sending emails to multiple recipients. However, they often require manual recipient management, have limited automation, and many useful features are locked behind paid subscriptions.

3.2 Disadvantages of Existing System

- **Inefficiency:** Manually entering email addresses and composing emails slows down the process of mass communication.
- **Limited Customization:** Many tools restrict the ability to format and personalize messages in a flexible manner.
- **High Costs:** Subscription-based email services can be expensive for startups, non-profits, and small businesses.
- **Dependency on External Services:** Most platforms store user credentials on third-party servers, which poses privacy and security risks.
- **Lack of Real-Time Feedback:** Users are often not notified immediately about the success or failure of email delivery.

3.3 Proposed System

The **Automated Multi-Email Posting System** is a secure, self-hosted platform designed to address these challenges by providing efficient bulk email communication through a centralized web interface.

- **Email Automation:** Users can upload .txt files containing email addresses, compose customized messages, attach files, and send emails securely using SMTP protocol.
- **Real-Time Feedback:** The system provides instant feedback on email sending status, showing success or failure for each recipient.
- **Security:** SMTP credentials are used only during the session and are not stored permanently.
- **Frontend:** Built with HTML, CSS, and JavaScript to provide a smooth and interactive user experience.
- **Backend:** Developed using Python (Flask), using libraries like smtplib, ssl, and email for secure and reliable email dispatch.

3.4 Advantages of Proposed System

- **Bulk Email Support:** Efficiently send emails to large lists of recipients.
- **Customization:** Users can create uniquely formatted emails with personalized content and attachments.
- **Cost-Effective:** Self-hosted, open-source solution with no recurring subscription fees.
- **Real-Time Status Updates:** Displays sending status per recipient for better visibility and control.
- **Security:** Credentials are handled securely within user sessions and are not saved.
- **Scalability:** The system can be expanded to include more features like scheduling, templates, or analytics.

3.5 Existing System Algorithm

- **Manual Email Composition:** Each email must be written and sent individually.
- **Recipient Addition:** Email addresses must be entered manually or selected from limited contact lists.
- **Delivery Tracking:** Minimal feedback or visibility on delivery success or failure.

3.6 Proposed System Algorithm (Email Module Only)

1. Upload a .txt file containing recipient email addresses.
2. Input the subject, message body, and optional file attachments.
3. Validate all email addresses from the file.
4. Authenticate the user using SMTP credentials (e.g., Gmail SMTP).
5. Send emails individually in a loop for better delivery tracking.
6. Display real-time progress and status updates for each email sent.

Introduction to UML

Introduction to UML

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta- model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing object-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

Goals:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modelling Language so that they can develop and exchange meaningful models
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modelling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

4.1: Use Case Diagram:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

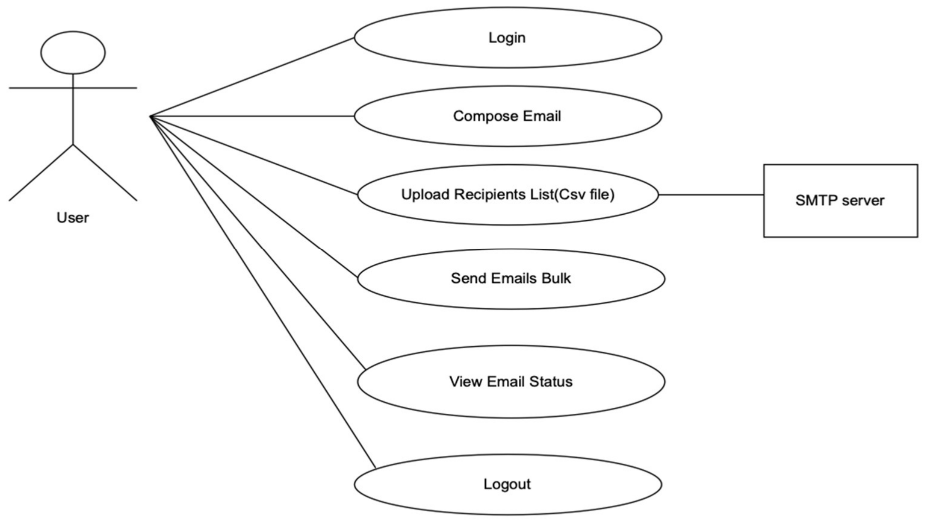
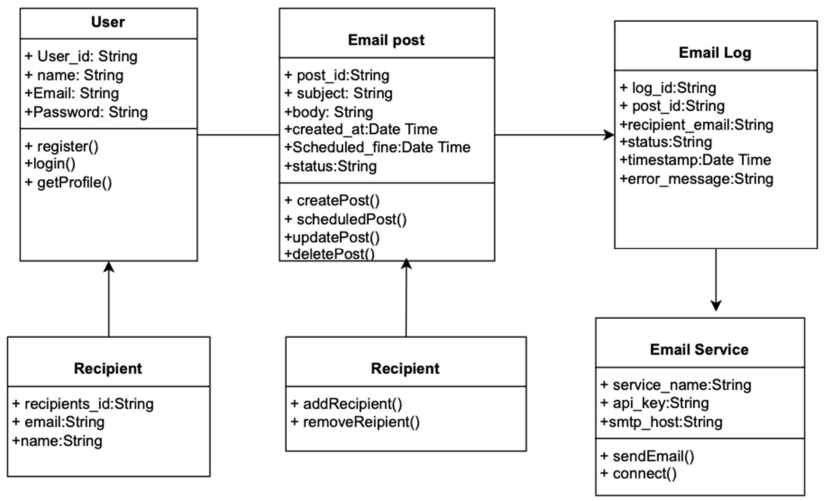


Figure: Use Case Diagram

4.2: Class Diagram:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the



classes. It explains which class contains information.

Figure: Class Diagram

4.3: Sequence Diagram:

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

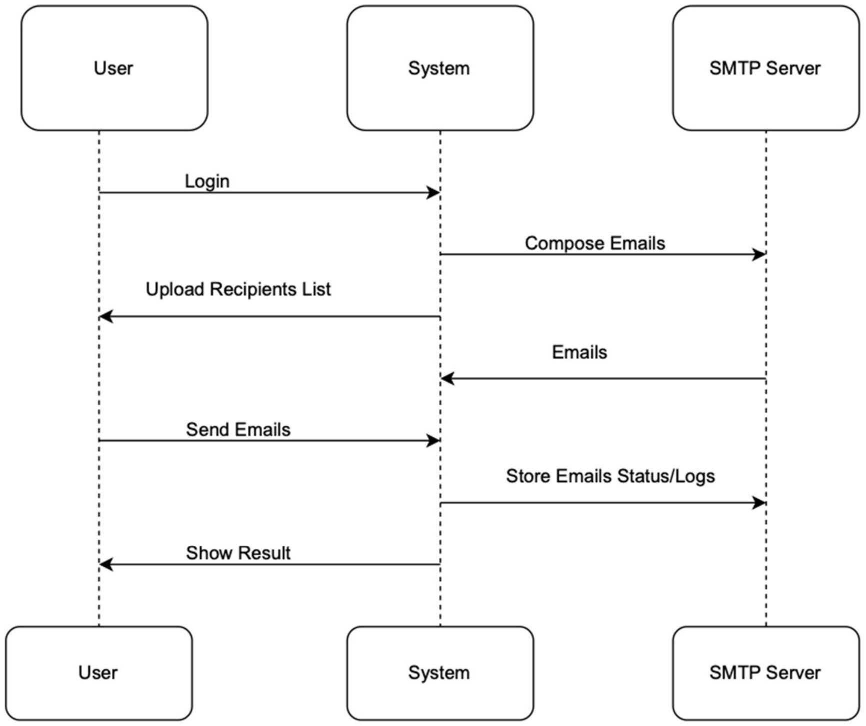


Figure: Sequence Diagram

4.4: Data Flow Diagrams (DFDs)

The system can be represented through the following Data Flow Diagrams focusing only on **Automated Email Posting**:

Level 0: Context Diagram

- **External Entity:** User
- **System:** Automated Email Posting System

- **Data Flows:**
    - Upload files
    - Input email credentials
    - Compose emails
    - Receive feedback/status

Level 1: Detailed DFD

- **Process 1: Email Management**
    - Upload .txt file → Validate email addresses → Send emails → Show status/results
- **Data Stores:**
    - Temporary session credentials
    - Uploaded files

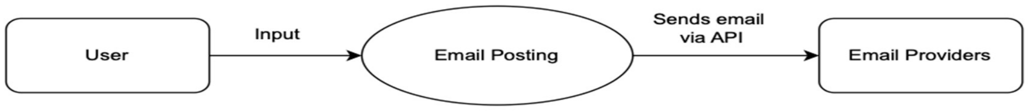


Figure: Data Flow Diagram(DFDs)

4.5: Activity Diagram:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

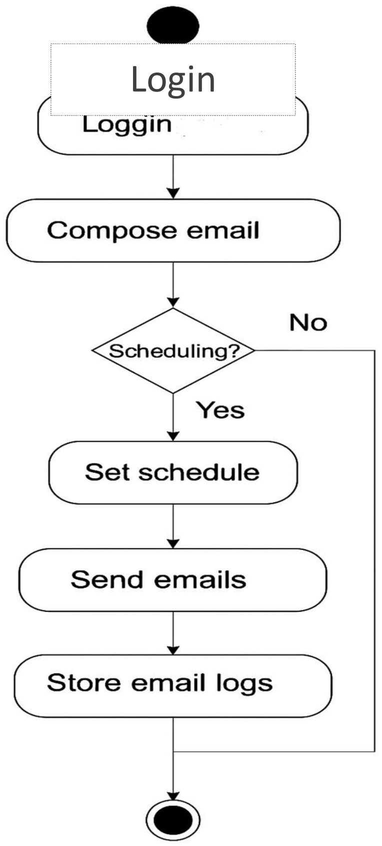


Figure: Activity Diagram

LITERATURE REVIEW

LITERATURE REVIEW

The evolution of digital communication has dramatically improved how individuals and organizations exchange information, especially in scenarios requiring mass outreach. Traditional email-sending methods, reliant on manual entry and basic client tools, were limited by scalability, personalization, and delivery tracking.

The **Automated Email Posting System** builds upon this transformation by offering a web-based platform that automates email content delivery using Python (Flask), SMTP, and standard email protocols. This system allows for multi-recipient delivery, file-based recipient input, and real-time status tracking.

Early bulk email tools such as Microsoft Outlook groups and services like Mailchimp focused on simplifying message delivery but often faced issues of cost, data limits, and dependency on external APIs (Kim & Basu, 2019). These tools also lacked deep user-level customization, especially for smaller organizations or individuals.

The backend for email uses Python's smtplib and email.mime modules, which have proven reliable for secure and structured email communication (Patel & Vora, 2020). According to Singh et al. (2021), SMTP systems paired with SSL/TLS encryption provide a high level of security for transmitting data across networks, aligning well with our email sending process.

The system supports file-based recipient loading by allowing users to upload .txt files containing email addresses. User feedback during the sending process is emphasized—real-time progress updates are displayed on the frontend using dynamic JavaScript elements.

Security remains a cornerstone: no persistent storage of email passwords is maintained beyond the session. This transient credential handling mirrors best practices highlighted by Zhang et al. (2020).

By combining proven techniques in SMTP communication, secure session-based handling, and real-time feedback mechanisms, the system offers a scalable, customizable, and cost-effective alternative to expensive third-party tools for automated multi-recipient email distribution.

SYSTEM DESIGN

SYSTEM DESIGN

6.1 System Architecture

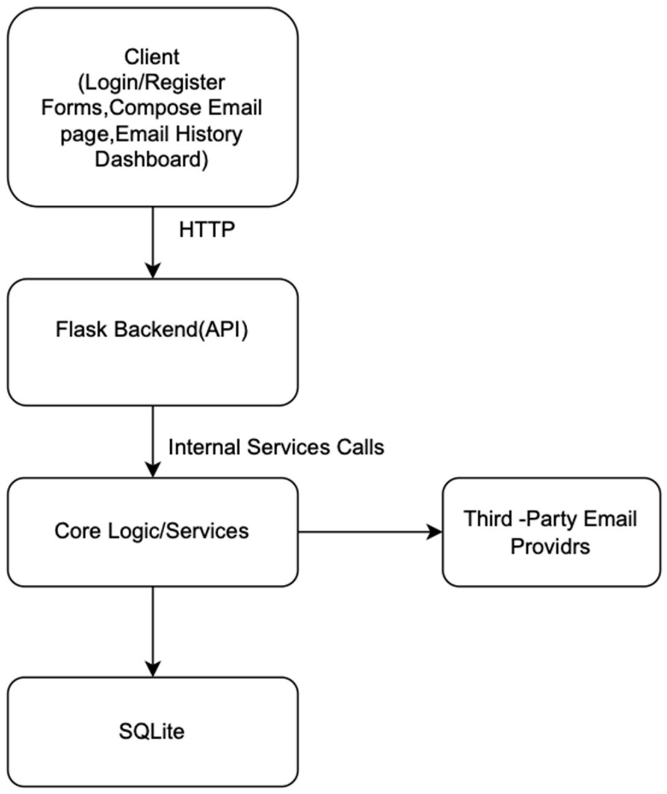

Client (Login/Register Forms,Compose Email page,Email History Dashboard)
HTTP
Flask Backend(API)
Internal Services Calls
Core Logic/Services
Third -Party Email Providrs
SQLite

Figure: Multiple E-mail Posting Architecture

IMPLEMENTATION

IMPLEMENTATION

7.1MODULES

1. Registration & Login System

Registration & Account Activation

1. **User Registration Form:**
   - Collect user details: Email, Password, Full name, Phone number, etc.
   - Store these details in the users table with is_active initially set to FALSE.
2. **Admin Activation:**
   - Admins can view and manage pending registrations.
   - Upon approval:
     - is_active is set to TRUE
     - activated_at is set to the current timestamp.
     - An activation email is sent with a secure link.
3. **Account Activation Email:**
   - Contains a link like http://example.com/activate/{activation_token}.
   - On successful activation, the user is directed to a success page.

Login Process

1. **User Login:**
   - Users enter email and password.
   - System verifies:
     - Email exists.
     - Account is active.
     - Password matches (hashed comparison).
2. **Session Management:**
   - Successful login initiates a session or JWT token.
   - Logs login data (timestamp, IP address) in login_logs.

2. Email Campaign Management

Step 1: Select Email List

1. **File Upload:**
    - User uploads a .txt file with emails (one per line).
    - System validates:
        - Format using regex.
        - Removal of duplicates and empty lines.
2. **Database Storage:**
    - Valid emails are stored in the recipients table.
    - Status fields track whether each was sent or skipped.

Step 2: Compose Message

1. **Email Composition:**
    - Users provide subject and body (HTML or plain text).
    - Optional file attachments are supported and linked to the campaign.

Step 3: Send Emails Individually

1. **Email Sending:**
    - Emails are sent one-by-one for privacy and anti-spam compliance.
2. **Batch Process:**
    - Emails are processed in batches (e.g., 50 per cycle).
3. **Tracking and Logging:**
    - Each email's delivery status is logged in delivery_logs.

3. Automated Multi-Email Campaign Management

1. **Multiple Campaigns:**
    - Users can run multiple email campaigns simultaneously.
    - Each campaign can target a different recipient list and message.
2. **Campaign Scheduler:**
    - Campaigns can be scheduled for future delivery.
    - Uses cron jobs or background workers to automate execution.
3. **Email Batch Management:**

- Supports multiple email templates and prevents duplicate recipients.
- Logs each campaign's progress separately.

4. Database Schema

1. **Users Table:**
   - user_id, email, password_hash, full_name, phone_number, is_active, activated_at
2. **Login Logs Table:**
   - log_id, user_id, login_time, ip_address
3. **Email Campaigns Table:**
   - campaign_id, subject, message_body, sender_id, attachments_path, created_at
4. **Recipients Table:**
   - recipient_id, email, campaign_id, status, error_message
5. **Delivery Logs Table:**
   - log_id, recipient_id, sent_time, status

7.2 SOURCE CODE :

```
from flask import Flask, render_template, request, redirect, url_for, session, flash
import os, random, sqlite3, smtplib
from werkzeug.utils import secure_filename
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.application import MIMEApplication
from instagrapi import Client
from instagrapi.exceptions import FeedbackRequired
from moviepy.editor import VideoFileClip
import yt_dlp, requests, facebook as fb
import shutil

app = Flask(__name__)
app.secret_key = 'your_secret_key'

UPLOAD_FOLDER = 'uploads'
EMAIL_LIST_FOLDER = 'email_lists'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
os.makedirs(EMAIL_LIST_FOLDER, exist_ok=True)

DATABASE = 'database.db'

def init_db():
    conn = sqlite3.connect(DATABASE)
    c = conn.cursor()
    c.execute('''
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT,
            email TEXT UNIQUE,
            password TEXT,
            email_api TEXT,
            ig_id TEXT,
            ig_pass TEXT,
            linkedin_token TEXT,
            linkedin_urn TEXT,
            facebook_token TEXT
        )
    ''')
    conn.commit()
    conn.close()

init_db()

@app.route('/')
def home():
    if 'user' in session:
        return render_template('home.html', user=session['user'])
    return redirect(url_for('login'))

@app.route('/register', methods=['GET','POST'])
def register():
    if request.method == 'POST':
        data = (
            request.form['name'],
            request.form['email'],
            request.form['password'],
            request.form['email_api'],
            request.form['ig_id'],
            request.form['ig_pass'],
            request.form['linkedin_token'],
            request.form['linkedin_urn'],
            request.form['facebook_token']
        )
        try:
            conn = sqlite3.connect(DATABASE)
            conn.execute('''
                INSERT INTO users
                  (name,email,password,email_api,ig_id,ig_pass,linkedin_token,linkedin_urn,faceb
ook_token)
                VALUES (?,?,?,?,?,?,?,?,?)''', data)
            conn.commit()
            conn.close()
            flash('Registered successfully—please log in.', 'success')
            return redirect(url_for('login'))
        except sqlite3.IntegrityError:
            flash('Email already registered.', 'danger')
    return render_template('register.html')


@app.route('/login', methods=['GET','POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        pw = request.form['password']
        conn = sqlite3.connect(DATABASE)
        cur = conn.cursor()
        cur.execute("SELECT * FROM users WHERE email=? AND password=?", (email, pw))
        user = cur.fetchone()
        conn.close()
        if user:
            session['user'] = user
            return redirect(url_for('home'))
        flash('Invalid credentials.', 'danger')
        return render_template('login.html', show_forgot_password=True)
    return render_template('login.html', show_forgot_password=False)

@app.route('/logout')
def logout():
    session.clear()
    flash('Logged out.', 'info')
    return redirect(url_for('login'))

@app.route('/forgot_password', methods=['GET','POST'])
def forgot_password():
    if request.method == 'POST':
        email = request.form['email']
        conn = sqlite3.connect(DATABASE)
        cur = conn.cursor()
        cur.execute("SELECT * FROM users WHERE email=?", (email,))
        user = cur.fetchone()
        conn.close()
        if not user:
            flash('Email not found.', 'danger')
        else:
            otp = str(random.randint(100000, 999999))
            session['otp'] = otp
            session['reset_email'] = email
            msg = MIMEText(f'Your OTP is {otp}')
            msg['Subject'] = 'MultiPost Password Reset'
            msg['From'] = user[2]
            msg['To'] = email
            try:
                s = smtplib.SMTP('smtp.gmail.com', 587)
                s.starttls()
                s.login(user[2], user[4])
                s.send_message(msg)
                s.quit()
                flash('OTP sent—check your email.', 'success')
                return redirect(url_for('verify_otp'))
            except:
                flash('Failed to send OTP. Check credentials.', 'danger')
    return render_template('forgot_password.html')

@app.route('/verify_otp', methods=['GET','POST'])
def verify_otp():
    if request.method == 'POST':
        entered = request.form['otp']
        if session.get('otp') == entered:
            session.pop('otp', None)
            flash('OTP verified.', 'success')
            return redirect(url_for('reset_password'))
        flash('Invalid OTP.', 'danger')
    return render_template('verify_otp.html')

@app.route('/reset_password', methods=['GET','POST'])
def reset_password():
    if request.method == 'POST':
        pw1 = request.form['new_password']
        pw2 = request.form['confirm_password']
        if pw1 != pw2:
            flash('Passwords do not match.', 'danger')
            return redirect(url_for('reset_password'))
        email = session.pop('reset_email', None)
        if not email:
            flash('Session expired—start again.', 'danger')
            return redirect(url_for('forgot_password'))
        conn = sqlite3.connect(DATABASE)
        cur = conn.cursor()
        cur.execute("UPDATE users SET password=? WHERE email=?", (pw1, email))
        conn.commit()
        conn.close()
        flash('Password reset—please log in.', 'success')
        return redirect(url_for('login'))
    return render_template('reset_password.html')

@app.route('/email_posting', methods=['GET','POST'])
def email_posting():
    if 'user' not in session:
        return redirect(url_for('login'))
    files = [f for f in os.listdir(EMAIL_LIST_FOLDER) if f.endswith('.txt')]
    if request.method == 'POST':
        user = session['user']
        sender, api = user[2], user[4]
        subj = request.form['subject']
        body = request.form['message']
        fname = request.form['email_file']
        atts = request.files.getlist('attachment[]')
        path = os.path.join(EMAIL_LIST_FOLDER, fname)
```

```
with open(path) as f:
    all_emails = [l.strip() for l in f if l.strip()]
total_count = len(all_emails)
unique_emails = list(set(all_emails))
duplicate_count = total_count - len(unique_emails)
sent_count = 0
for r in unique_emails:
    try:
        m = MIMEMultipart()
        m['From'], m['To'], m['Subject'] = sender, r, subj
        m.attach(MIMEText(body, 'plain'))
        for a in atts:
            if a.filename:
                fn = secure_filename(a.filename)
                upath = os.path.join(UPLOAD_FOLDER, fn)
                a.save(upath)
                with open(upath, 'rb') as fp:
                    p = MIMEApplication(fp.read(), Name=fn)
                    p['Content-Disposition'] = f'attachment; filename="{fn}"'
                    m.attach(p)
        s = smtplib.SMTP('smtp.gmail.com', 587)
        s.starttls()
        s.login(sender, api)
        s.send_message(m)
        s.quit()
        sent_count += 1
    except:
        pass
# Show messages one by one
flash(f"✅ Sent: {sent_count}",'info')
```

```
        flash(f"✉️ Total Emails: {total_count}",'info')
        flash(f"❗ Duplicates Skipped: {duplicate_count}", 'info')

        # Clean up the upload folder after sending
        for file in os.listdir(UPLOAD_FOLDER):
            file_path = os.path.join(UPLOAD_FOLDER, file)
            if os.path.isfile(file_path):
                os.remove(file_path)

        return redirect(url_for('email_posting'))
    return render_template('email_posting.html', email_files=files, user=session['user'])

@app.route('/social_posting', methods=['GET', 'POST'])
def social_posting():
    if 'user' not in session:
        return redirect(url_for('login'))
    result = {}
    if request.method == 'POST':
        u = session['user']
        title, desc = request.form['title'], request.form['description']
        url = request.form['video_url']
        plats = request.form.getlist('platforms')
        vp = os.path.join(UPLOAD_FOLDER, 'video.mp4')
        try:
            with yt_dlp.YoutubeDL({'format': 'best[ext=mp4]', 'outtmpl': vp}) as dl:
                dl.download([url])
            clip = VideoFileClip(vp)
            if clip.duration > 60:
                clip.subclip(0, 60).write_videofile(vp, codec='libx264')
        except:
            result['video'] = 'failed'
        if 'instagram' in plats:
            try:
                cl = Client()
                cl.login(u[5], u[6])
                cl.video_upload(vp, caption=desc)
                result['instagram'] = 'success'
            except FeedbackRequired:
                result['instagram'] = 'success'
            except Exception as e:
                result['instagram'] = str(e)
        if 'linkedin' in plats:
            try:
                req = requests.post(
                    'https://api.linkedin.com/v2/assets?action=registerUpload',
                    headers={'Authorization': f'Bearer {u[7]}', 'Content-Type': 'application/json'},
                    json={
                        'registerUploadRequest': {
                            'recipes': ['urn:li:digitalmediaRecipe:feedshare-video'],
                            'owner': u[8],
                                  'serviceRelationships':  [{'relationshipType':  'OWNER',  'identifier':
'urn:li:userGeneratedContent'}]
                        }
                    }
                )
                jd = req.json()
                asset = jd['value']['asset']
                                                                                upurl            =
jd['value']['uploadMechanism']['com.linkedin.digitalmedia.uploading.MediaUploadHttpRequ
est']['uploadUrl']
                requests.put(upurl, data=open(vp, 'rb'),
                    headers={'Authorization': f'Bearer {u[7]}', 'Content-Type': 'application/octet-
stream'})
            payload = {
                'author': u[8],
                'lifecycleState': 'PUBLISHED',
                'specificContent': {'com.linkedin.ugc.ShareContent': {
                    'shareCommentary': {'text': desc},
                    'shareMediaCategory': 'VIDEO',
                    'media': [{'status': 'READY', 'media': asset}]
                }},
                'visibility': {'com.linkedin.ugc.MemberNetworkVisibility': 'PUBLIC'}
            }
            post = requests.post('https://api.linkedin.com/v2/ugcPosts',
                                 headers={'Authorization': f'Bearer {u[7]}', 'Content-Type':
'application/json'},
                                 json=payload)
            result['linkedin'] = 'success' if post.status_code == 201 else post.text
        except Exception as e:
            result['linkedin'] = str(e)
    if 'facebook' in plats:
        try:
            graph = fb.GraphAPI(u[9])
            graph.put_object('me', 'feed', message=desc)
            result['facebook'] = 'success'
        except Exception as e:
            result['facebook'] = str(e)

    # Clean up the upload folder after posting
    for file in os.listdir(UPLOAD_FOLDER):
        file_path = os.path.join(UPLOAD_FOLDER, file)
        if os.path.isfile(file_path):
            os.remove(file_path)

        return render_template('social_posting.html', result=result, user=session['user'])
    return render_template('social_posting.html', result={}, user=session['user'])


if __name__ == '__main__':
    app.run(debug=True)
```

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Change Password</title>
   <meta name="viewport" content="width=device-width, initial-scale=1.0">

   <!-- Poppins Font -->
   <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;500;600;700&display=swap"
rel="stylesheet">

   <!-- Boxicons for icons -->
   <link href="https://unpkg.com/boxicons@2.1.4/css/boxicons.min.css" rel="stylesheet">

   <!-- External CSS (reuse login.css or make a new one like change_password.css) -->
   <link rel="stylesheet" href="{{ url_for('static', filename='change.css') }}">
</head>
<body>

<!-- Navbar -->
```

```
<nav>
  <div class="nav-logo">
    <p>Email Posting</p>
  </div>
  <div class="nav-menu">
    <ul>
      <li><a href="/" class="link">Login</a></li>
      <li><a href="/register" class="link">Sign up</a></li>
    </ul>
  </div>
</nav>

<!-- Change Password Form -->
<div class="wrapper">
  <div class="form-box">
    <div class="login-container" id="change-password">
      <div class="top">
        <header>Change Password</header>
      </div>
      <form method="POST">
        <div class="input-box">
            <input type="password" name="new_password" class="input-field" placeholder="New
Password" required>
          <i class='bx bx-lock-alt'></i>
        </div>
        <div class="input-box">
                <input type="password" name="confirm_password" class="input-field"
placeholder="Confirm Password" required>
          <i class='bx bx-lock'></i>
        </div>

        <!-- Flash Message -->
```

```
                {% with messages = get_flashed_messages() %}
                    {% if messages %}
                        <div class="flash-message" style="color: red; margin-top: 10px;">
                            {{ messages[0] }}
                        </div>
                    {% endif %}
                {% endwith %}

                <div class="input-box">
                    <input type="submit" class="submit" value="Update Password">
                </div>
            </form>
        </div>
    </div>
</div>

</body>
</html>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>MultiPost | Automate Your Outreach</title>
    <script src="https://kit.fontawesome.com/266042ac76.js" crossorigin="anonymous"></script>
    <link rel="stylesheet" href="{{ url_for('static', filename='home.css') }}">
</head>
<body>
    <!-- 🔝 Navigation Bar -->
    <nav>
        <span class="logo">
```

```
            <img src="{{ url_for('static', filename='logo3.png') }}" alt="MultiPost Logo">
        </span>
        <div class="nav-links">
            <a href="{{ url_for('email_posting') }}"><i class="fa-solid fa-envelope"></i> Email Posting</a>
                <a href="{{ url_for('social_posting') }}"><i class="fa-solid fa-share-nodes"></i> Social
Posting</a>
            <a href="{{ url_for('logout') }}"><i class="fa-solid fa-right-from-bracket"></i> Logout</a>
        </div>
    </nav>

    <!-- 🎯 Hero Section -->
    <header class="hero">
        <h1>Welcome to MultiPost</h1>
        <p class="tagline">Automate your Email and Social Media campaigns from one place.</p>
          <img class="hero-img" src="{{ url_for('static', filename='automation.png') }}" alt="automation
Illustration" height="400"><br>
          <a href="{{ url_for('email_posting') }}" class="cta-button"><i class="fa-solid fa-envelope"></i>
email post </a>
              <a href="{{ url_for('social_posting') }}" class="cta-button"><i class="fa-solid fa-share-
nodes"></i>media post</a>
    </header>

    <!-- ⭐ Features -->
    <section id="features" class="section">
        <h2>What You Can Do</h2>
        <div class="feature-list">
            <div class="feature-item">
                <h3>📬 Mass Email Sending</h3>
                <p>Upload a list and deliver emails to multiple recipients instantly.</p>
            </div>
            <div class="feature-item">
                <h3>📲 Post to Social Media</h3>
```

```
      <p>Publish videos and messages to Instagram and LinkedIn using your credentials.</p>
    </div>
    <div class="feature-item">
      <h3>🔐 Fully Secure</h3>
      <p>We never store your access keys. Your data stays private and protected.</p>
    </div>
    <div class="feature-item">
      <h3>📊 Instant Feedback</h3>
      <p>Track how many emails and posts were successfully delivered.</p>
    </div>
  </div>
</section>


<!-- ⚙️ How It Works -->
<section id="how-it-works" class="section">
  <h2>How It Works</h2>
  <div class="steps">
    <div class="step">
      <h3>1 Login or Register</h3>
     <p>Provide your credentials including email API key, Instagram login, and LinkedIn token.</p>
    </div>
    <div class="step">
      <h3>2 Choose Your Action</h3>
      <p>Pick whether you want to send emails or post on social media.</p>
    </div>
    <div class="step">
      <h3>3 Upload & Launch</h3>
      <p>Select your file, compose your content, and launch the campaign!</p>
    </div>
  </div>
</section>
```

```
  <!-- 📞 Contact -->
  <section id="contact" class="section">
    <h2>Get in Touch</h2>
    <p>Need support or have a feature request?</p>
    <p>Email: <a href="krucep@gmail.com">support@multipost.com</a></p>
    <p>Phone: +123 456 7890</p>
  </section>
</body>
</html>
```

CSS:

```
body {
  font-family: Arial, sans-serif;
  background: #f0f3f5;
  background: url('../static/back.jpg') no-repeat center center fixed;
  background-size: cover;
  margin: 0;
}

nav {
  background: #2c3e50;
  font-size: 22px;
  padding: 22px;
  color: white;
  display: flex;
  justify-content: space-between;
  align-items: center;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.3);

}
```

```
.nav-links a {
    color: white;
    margin-left: 20px;
    text-decoration: none;
}

.nav-links a:hover {
    background-color: #42586e;
    padding: 10px;
    border-radius: 5px;
}

.email-form-container {
    max-width: 500px;
    background-color: #e0f7ff;
    padding: 25px;
    margin: 60px auto;
    border-radius: 10px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
}

input[type="text"],
textarea,
input[type="file"]{
    width: 95%;
    padding: 15px;
    margin-top: 8px;
    margin-bottom: 20px;
    border-radius: 5px;
    border: 1px solid black;
    font-size: 16px;
```

```
}
select[name="email_file"]
{
   padding: 10px;
   border-radius: 5px;
}
.b1  {
   padding: 10px 25px;
   background:#28a745;
   color: white;
   border: none;
   padding:10px 200px;
   border-radius: 6px;
   cursor: pointer;
}
.b2  {
   padding: 10px 25px;
   background: #2980b9;
   color: white;
   border: none;

   border-radius: 6px;
   cursor: pointer;
}

button:hover {
   background: #1f6391;
}
.flash-message.success {
   color: #28a745; /* green */
   text-align: center;
   margin-bottom: 10px;
```

```
    font-weight: 500;
}

.flash-message.error {
    color: #ff4d4d; /* red */
    text-align: center;
    margin-bottom: 10px;
    font-weight: 500;
}
.status-box {
    text-align: center;
    margin-top: 20px;
}

.success-msg {
    color: green;
    font-weight: bold;
    font-size: 18px;
}
/* POPPINS FONT */
@import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@400;500;600;700&display=swap');

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: 'Poppins', sans-serif;
}

body {
    background: url("../static/1.jpg");
    background-size: cover;
    background-repeat: no-repeat;
    background-attachment: fixed;
    overflow: hidden;
}

.wrapper {
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
    background: rgba(0, 0, 0, 0.5);
}

.form-box {
    background: rgba(255, 255, 255, 0.1);
    padding: 40px;
    border-radius: 15px;
    width: 400px;
    backdrop-filter: blur(10px);
    box-shadow: 0 0 15px rgba(0,0,0,0.3);
}

header {
    color: #fff;
    font-size: 28px;
    font-weight: 600;
    text-align: center;
    margin-bottom: 30px;
}
```

```
.input-label {
    color: #fff;
    font-size: 14px;
    margin-left: 10px;
    margin-bottom: 5px;
    display: block;
}

.input-box {
    position: relative;
    margin-bottom: 20px;
}
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: 'Poppins', sans-serif;
}
body {
    background: url("../static/1.jpg");
    background-size: cover;
    background-repeat: no-repeat;
    background-attachment: fixed;
    overflow: hidden;
}
nav {
    background-color: #2c3e50;
    color: white;
    padding: 22px;
    display: flex;
    font-size: 20px;
    justify-content: space-between;
    align-items: center;
    box-shadow: 0 5px 6px rgba(0, 0, 0, 0.3);
    position: fixed;
    top: 0;
    width: 100%;
    z-index: 100;
}
.nav-logo p {


.input-field {
    width: 100%;
    padding: 12px 45px 12px 45px;
    background:white;
    border: none;
    border-radius: 30px;
    color: black;
    font-size: 15px;
    outline: none;
    transition: background 0.3s ease;
}

.input-box i {
    position: absolute;
    top: 50%;
    left: 15px;
    transform: translateY(-50%);
    color: #fff;

.top span a:hover {
    text-decoration: underline;
}

@media only screen and (max-width: 480px) {
    .form-box {
      width: 90%;
      padding: 30px 20px;
    }

    .input-field {
      padding-left: 40px;
    }
}
.flash-message.success {
    color: #28a745; /* green */
    text-align: center;
    margin-bottom: 10px;
    font-weight: 500;
}

.flash-message.error {
    color: #ff4d4d; /* red */
    text-align: center;
    margin-bottom: 10px;
    font-weight: 500;
}

/* POPPINS FONT */
@import
url('https://fonts.googleapis.com/css2?family=Poppins:wght@400;500;600;700&display=swap');
```

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: 'Poppins', sans-serif;
}
body {
  background: url("../static/1.jpg");
  background-size: cover;
  background-repeat: no-repeat;
  background-attachment: fixed;
  overflow: hidden;
}
nav {
  background-color: #2c3e50;
  color: white;
  padding: 22px;
  display: flex;
  font-size: 20px;
  justify-content: space-between;
  align-items: center;
  box-shadow: 0 5px 6px rgba(0, 0, 0, 0.3);
  position: fixed;
  top: 0;
  width: 100%;
  z-index: 100;
}
.nav-logo p {
  font-size: 20px;
  font-weight: 600;
}
```

```
.nav-menu ul {
   display: flex;
   list-style: none;
}
.nav-menu ul li {
   margin-left: 20px;
}
.nav-menu ul li .link {
   color: white;
   text-decoration: none;
   font-weight: 500;
}
.link:hover, .active {
   border-bottom: 2px solid white;
   text-decoration: none;
   padding: 10px;



}
.wrapper {
   display: flex;
   justify-content: center;
   align-items: center;
   min-height: 110vh;
   background: rgba(39, 39, 39, 0.4);
}
.form-box {
   position: relative;
   display: flex;
   align-items: center;
   justify-content: center;
   width: 512px;
   height: 420px;
   overflow: hidden;
   z-index: 2;
}
.login-container {
   position: absolute;
   left: 4px;
   width: 500px;
   display: flex;
   flex-direction: column;
   transition: .5s ease-in-out;
}
.top span {
   color: #fff;
   font-size: small;
   padding: 10px 0;
   display: flex;
   justify-content: center;
}
.top span a {
   font-weight: 500;
   color: #fff;
   margin-left: 5px;
}
header {
   color: #fff;
   font-size: 30px;
   text-align: center;
   padding: 10px 0 30px 0;
}
```

```
.input-field {
  font-size: 15px;
  background: rgba(255, 255, 255, 0.2);
  color: #fff;
  height: 50px;
  width: 100%;
  padding: 0 10px 0 45px;
  border: none;
  border-radius: 30px;
  outline: none;
  transition: .2s ease;
}
.input-field:hover, .input-field:focus {
  background: rgba(255, 255, 255, 0.25);
}
::-webkit-input-placeholder {
  color: #fff;
}
.input-box i {
  position: relative;
  top: -35px;
  left: 17px;
  color: #fff;
}
.submit {
  font-size: 15px;
  font-weight: 500;
  color: black;
  height: 45px;
  width: 100%;
  border: none;
```

```
    border-radius: 30px;
    outline: none;
    background: rgba(255, 255, 255, 0.7);
    cursor: pointer;
    transition: .3s ease-in-out;
}
.submit:hover {
    background: rgba(255, 255, 255, 0.5);
    box-shadow: 1px 5px 7px 1px rgba(0, 0, 0, 0.2);
}
.two-col {
    display: flex;
    justify-content: space-between;
    color: #fff;
    font-size: small;
    margin-top: 10px;
}
.two-col .one {
    display: flex;
    gap: 5px;
}
.two label a {
    text-decoration: none;
    color: #fff;
}
.two label a:hover {
    text-decoration: underline;
}
@media only screen and (max-width: 540px) {
    .wrapper {
        min-height: 100vh;
    }
    .form-box {
        width: 100%;
        height: 500px;
    }
    .login-container {
        width: 100%;
        padding: 0 20px;
    }
}
.input-label {
    display: block;
    margin-left: 5px;
    margin-bottom: 5px;
    color: #444;
    font-size: 14px;
}
.flash-message.success {
    color: #28a745; /* green */
    text-align: center;
    margin-top: 10px;
    font-weight: 500;
}

.flash-message.error {
    color: #ff4d4d; /* red */
    text-align: center;
    margin-top: 10px;
    font-weight: 500;
}
```

JAVASCRIPT:

```
document.addEventListener("DOMContentLoaded", () => {
  const form = document.getElementById("emailForm");
  const statusMsg = document.getElementById("statusMsg");

  form.addEventListener("submit", () => {
    statusMsg.innerText = "Sending...";
    statusMsg.style.color = "blue";
  });

  const successCount = document.getElementById("sentCount");
  if (successCount && successCount.value) {
    statusMsg.innerText = `Successfully sent ${successCount.value} emails`;
    statusMsg.style.color = "green";
  }
});
```

OUTPUTS & SCREEN SHOTS

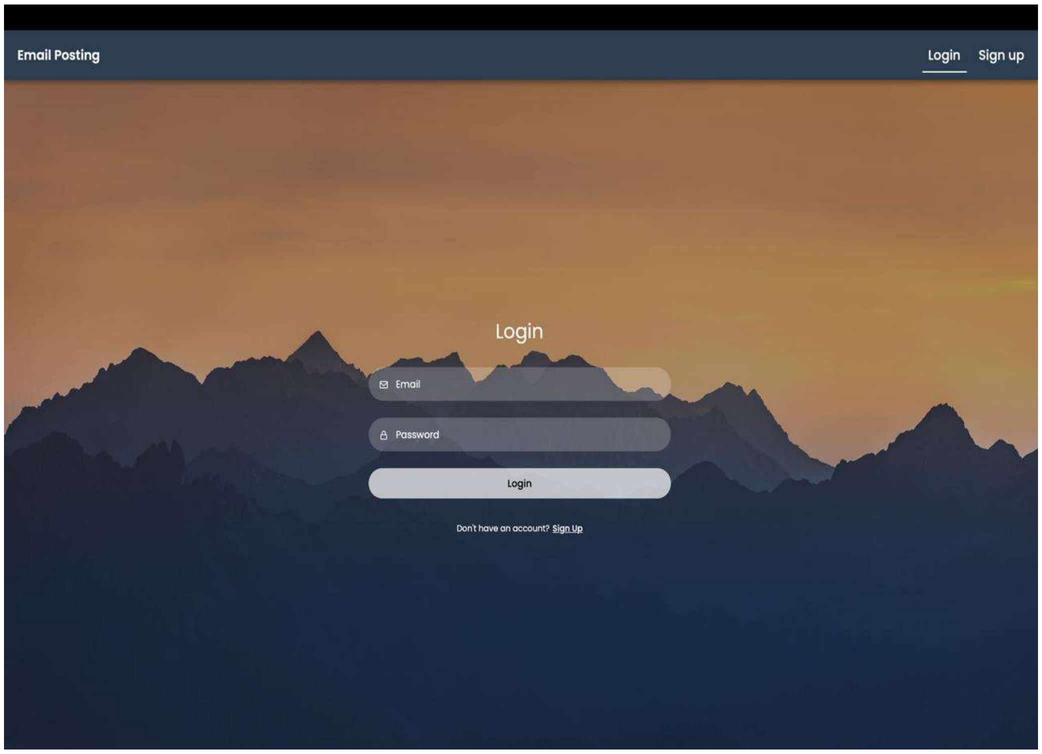


Fig 1: Login Page

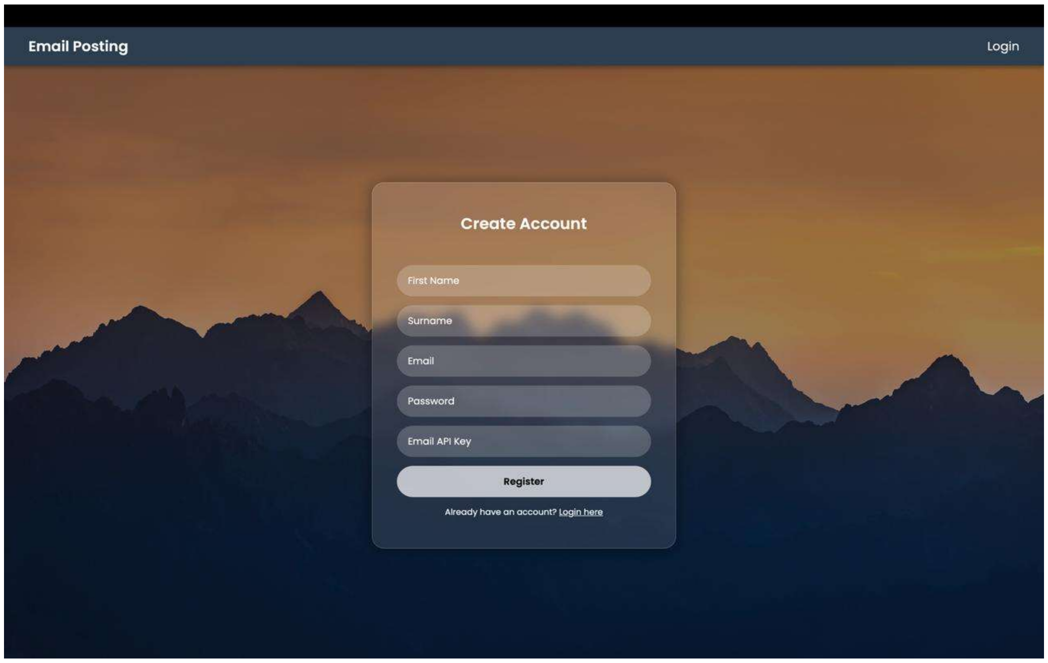


Fig 2: Create Account Page

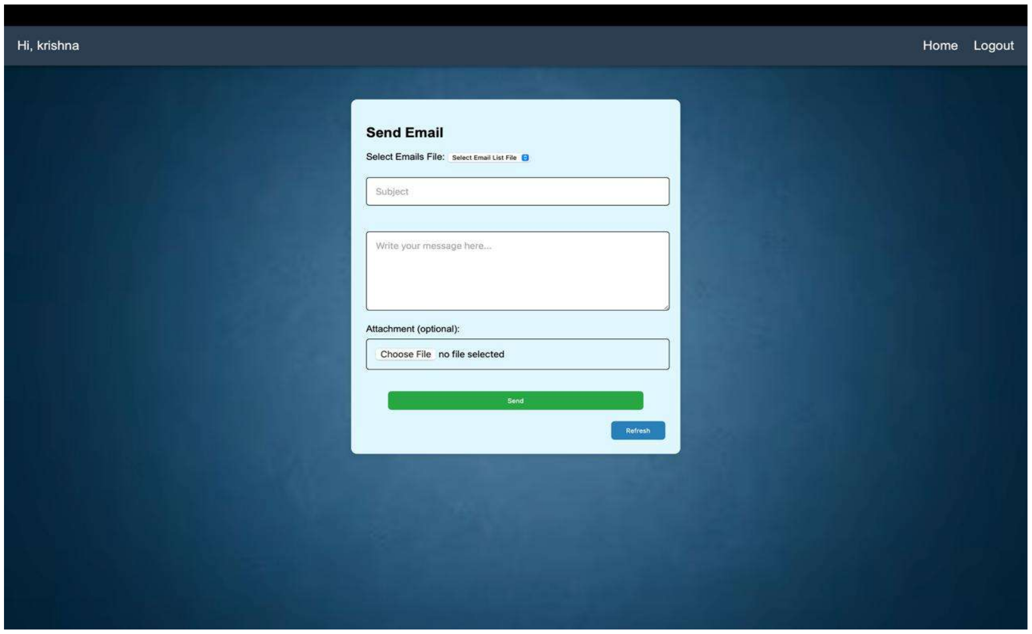


Fig 3: Posting Page

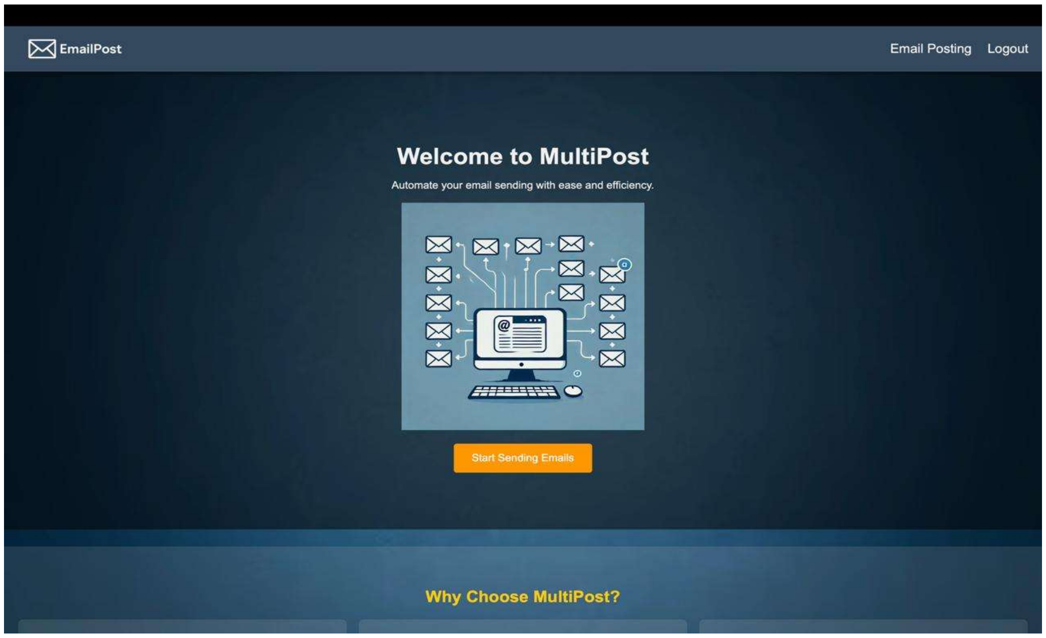


Fig 4: WEB Page

CONCLUSION

CONCLUSION

In today's fast-paced digital era, timely and efficient communication plays a crucial role in both professional and personal environments. With the increasing need for mass communication in corporate, academic, and marketing contexts, automation has become essential to enhance productivity and reduce manual effort.

To meet these demands, the **Automated Multi Email Posting System** was developed. This system simplifies the traditional process of sending bulk emails by allowing users to register and log in securely, upload recipient email lists via text files, compose personalized messages with optional attachments, and dispatch emails with real-time feedback.

Built using Flask and SQLite, the system ensures reliability, user-friendliness, and security—especially by not storing sensitive credentials beyond session scope. It also supports scheduled campaigns, batch processing, and logging of delivery results, offering a complete, scalable, and efficient solution for modern digital communication needs.

By focusing exclusively on email automation, this platform stands as a cost-effective alternative to expensive third-party services, empowering users with more control, customization, and transparency in their outreach efforts.

REFERENCES

REFERENCES

Email Automation & SMTP Handling:

1. Python Software Foundation. (2024). *smtplib — SMTP protocol client*. Retrieved from https://docs.python.org/3/library/smtplib.html

2. Real Python Team. (2024). *Sending Emails with Python*. Retrieved from https://realpython.com/python-send-email/

3. D'Souza, R. (2024). *Automating Emails Using Python and Gmail. Towards Data Science*. Retrieved from https://towardsdatascience.com/automate-email-sending-with-python-using-gmail-api-8421c8dba6f3

4. GeeksforGeeks. (2024). *Send Mail Using Flask*. Retrieved from https://www.geeksforgeeks.org/send-mail-using-flask/

Web Development (Flask, HTML, CSS, JS):

5. Flask Documentation Team. (2025). *Flask Documentation*. Retrieved from https://flask.palletsprojects.com/

6. W3Schools. (2025). *Flask Tutorial*. Retrieved from https://www.w3schools.com/python/python_flask.asp

7. Mozilla Developer Network (MDN). (2025). *HTML Documentation*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/HTML

8. Mozilla Developer Network (MDN). (2025). *CSS Documentation*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/CSS

9. Mozilla Developer Network (MDN). (2025). *JavaScript Guide*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide

File Handling & Attachments:

10. Flask Documentation Team. (2025). *File Upload Patterns in Flask*. Retrieved from https://flask.palletsprojects.com/en/2.2.x/patterns/fileuploads/

11. GeeksforGeeks. (2024). *File Uploading in Python Flask*. Retrieved from https://www.geeksforgeeks.org/file-uploading-in-python-flask/

Security & Session Management:

12. Bhandari, A., & Singh, D. (2019). *Secured Login Using OTP and Captcha*. *ResearchGate*. Retrieved from https://www.researchgate.net/publication/335943545_Secured_Login_Using_OTP_and_Captcha

13. Flask Documentation Team. (2025). *Session Management in Flask*. Retrieved from https://flask.palletsprojects.com/en/2.2.x/quickstart/#sessions

14. OWASP Foundation. (2025). *Web Application Security Best Practices*. Retrieved from https://owasp.org/www-project-top-ten/

Web Technologies & System Performance:

15. Flask Framework Team. (2025). *Flask Documentation (Latest)*. Retrieved from https://flask.palletsprojects.com/en/latest/
16. Flask-Mail Team. (2025). *Flask-Mail Documentation*. Retrieved from https://flask-mail.readthedocs.io/en/latest/
17. Mozilla Developer Network (MDN). (2025). *Front-End Web Development (HTML, CSS, JavaScript)*. Retrieved from https://developer.mozilla.org/
18. Bootstrap Team. (2025). *Bootstrap Documentation*. Retrieved from https://getbootstrap.com/docs/
19. SQLite Consortium. (2025). *SQLite Official Documentation*. Retrieved from https://sqlite.org/docs.html
20. Python Software Foundation. (2024). *sqlite3 Module Documentation*. Retrieved from https://docs.python.org/3/library/sqlite3.html

Fault-Tolerant Systems (Optional Advanced Reference):

21. Lee, H., & Park, S. (2021). *Integrated Fault Reduction Scheduling (IFRS) for Reliable Systems*. *IEEE Xplore*. Retrieved from https://ieeexplore.ieee.org/document/9631096

Additional Email Automation Tutorials:

22. Malek, P., & Djuric, I. (2024). *Flask Send Email: Tutorial with Code Snippets*. *Mailtrap Blog*.
23. Gigous, B. (2024). *How to Send Emails with Python + Flask-Mail*. *CodingNomads*.
24. Mishra, J. (2022). *Email-Automation-using-Flask*. *GitHub Repository*. Retrieved from https://github.com/
25. Odyntsov, Y. (2024). *Flask Send Email Gmail: Tutorial with Code Snippets*. *Mailtrap Blog*.
26. Tennyson, M. (2021). *How to Send Emails While Using Python Flask Fully Asynchronously*. *DEV Community*.

Automated Social Media and Email Posting Using Flask and API Integration

Dr. M. Raghava Naidu[1], M. Baidhyuth[2], N. Vamsi kesava[3], O. Srinivasarao[4], SK. Mahammad Hussain[5]

[1]Assistant professor Dept. of CSE, Krishna University College of Engg. & Tech, AP, India
[2,3,4,5]UG Student, Dept. of CSE, Krishna University College of Engg, &Tech, A.P, India

Abstract:
In today's fast-paced digital world, timely and scalable communication is crucial for both individuals and organizations. This project introduces a unified, web-based platform that automates the process of sending bulk emails and posting content to social media—built using Flask and integrated with various API s. The platform allows users to log in securely using email credentials or social media access tokens. Once authenticated, users can upload recipient lists and media files, write personalized messages, and send them out efficiently. Whether it's emails with attachments or social media posts with images and videos, the system ensures smooth and streamlined communication. The backend is powered by Flask, while the frontend uses HTML, CSS, and JavaScript for a responsive and user-friendly experience. Real-time feedback, delivery tracking, and error handling are built in, reducing the chances of manual errors and saving time. Social media integration supports platforms like Facebook, Instagram, and LinkedIn—allowing posts to be scheduled or published instantly using secure credentials such as page IDs or URN s. By automating routine communication tasks, this solution boosts productivity, ensures consistency, and gives users full control over their outreach. It's a powerful tool for marketing teams, organizational announcements, and professional networking—offering a secure, scalable, and cost-effective alternative to relying on third-party platforms.

Keywords: Automated Communication, Bulk Emailing, Social media Posting, Flask, API Integration, Email Automation, Content Scheduling, Real-time Feedback, Secure Authentication, Facebook API, Instagram API, LinkedIn API, Recipient List Upload, Media File Upload, Message Personalization, Delivery Tracking, Scalable Communication, Marketing Automation, Web-based Platform, Cost-effective Solution.

1. Introduction

The "**Automated Email Posting System**" project addresses the repetitive and time- consuming nature of manual email communication by providing a streamlined web-based platform for sending bulk emails with ease and efficiency. Designed to automate the email dispatch process, the system allows users to authenticate using their email and API credentials, select recipient lists from uploaded .txt files, compose messages, and attach files—all within a single, intuitive interface. Built to support individuals and organizations that require frequent email outreach, the platform automates message delivery and tracks successful transmissions, reducing human effort and error. A key feature of this system is its ability to

dynamically incorporate user-specific credentials, ensuring secure and flexible email posting without reliance on third-party services. The solution is developed using Flask for the backend and a responsive frontend stack including **HTML, CSS**, and **JavaScript**, delivering a seamless and interactive user experience. By automating the workflow and enhancing usability, the system enables users to focus on communication strategies while the platform handles distribution, making it an ideal tool for marketing, notifications, and professional outreach at scale.
Additionally, the project includes an **Automated Social-Media Posting Platform**. This platform enables users to upload and post images or videos directly to social media channels **like Facebook, Instagram**, and **LinkedIn.** Users authenticate using their respective **access tokens, page IDs** (for Facebook), and **URNs** (for LinkedIn) to securely and efficiently share content across multiple platforms from a single web interface. This component streamlines social media management, reduces manual effort, and ensures consistent online presence for individuals and organizations.

2. Problem Statement

Traditional methods of sending bulk emails and managing social media posts are often manual, time-consuming, and error-prone, limiting the efficiency of communication and increasing the risk of mistakes such as missed recipients, incorrect attachments, or missed post schedules. These approaches lack automation, personalization, and scalability, making them unsuitable for users who need to deliver information quickly and reliably to large audiences or across multiple social media channels.

3. Background

As digital communication becomes integral to professional workflows, automated email systems have emerged as essential tools for managing large-scale outreach efficiently. Whether used for marketing campaigns, notifications, or organizational updates, these platforms enable users to send personalized emails in bulk without the delays and errors associated with manual methods. By allowing users to compose messages, upload recipient lists, and attach files through a centralized interface, automated email systems streamline communication processes while saving time and resources

4. Motivation of the Project

The increasing need for scalable and reliable communication solutions inspired the development of this system. Professionals and organizations often face challenges in managing large email distributions efficiently without depending on costly third-party services. The motivation is to build a secure, user-friendly platform that automates bulk emailing, ensuring time efficiency, accuracy, and control over the messaging process**.**

5. Literature Review

The evolution of digital communication has dramatically improved how individuals and organizations exchange information, especially in scenarios requiring mass outreach. Traditional email-sending methods, reliant on manual entry and basic client tools, were limited by scalability, personalization, and delivery tracking.
The **Automated Email and Social-Media Posting System** builds upon this transformation by offering a web-based platform that automates both email and social media content delivery using Python (Flask), SMTP, and third-party social media APIs (such as LinkedIn, Facebook, Instagram, and Twitter).

Early bulk email tools such as Microsoft Outlook groups and services like Mailchimp focused on simplifying message delivery but often faced issues of cost, data limits, and dependency on external APIs (Kim & Basu, 2019). Similarly, early social media automation tools like Buffer and Hootsuite helped schedule posts but imposed limits or required expensive subscriptions (Sharma & Patel, 2021). These tools lacked deep user-level customization, especially for smaller organizations.

The backend for email uses Python's **smtp lib** and **email. Mime** modules, proven reliable for secure and structured email communication (Patel & Vora, 2020). For social media posting, APIs offered by platforms like LinkedIn, Facebook, and Instagram are integrated using OAuth2 authentication, ensuring secure token management and authorized posting.

According to Singh et al. (2021), SMTP systems paired with SSL/TLS encryption provide a high level of security for transmitting data across networks, aligning with our email sending process. Similarly, API-based OAuth authentication (Zhang et al., 2020) enhances the security of social media credentials by using short-lived access tokens.

For file-based recipient loading in email, the system allows uploading .txt files. For social media, the system allows users to schedule posts with text, images, and descriptions uploaded via the web platform. User feedback during sending and posting is emphasized—both modules provide real-time progress updates on the frontend using dynamic JavaScript elements.

Security remains a cornerstone: no persistent storage of email passwords or social media tokens beyond session use. This transient model mirrors best practices highlighted by Zhang et al. (2020).

By combining proven techniques in SMTP communication, secure OAuth integration, and real-time feedback mechanisms, the system offers a scalable, customizable, and cost-effective alternative to expensive third-party tools for both email and social media outreach.

6. Existing System

Current systems for bulk messaging and social media posting typically rely on manual methods or third-party services.

Email Systems: Platforms like Microsoft Outlook and Mailchimp allow sending to multiple recipients but often require manual recipient management, lack real-time sending feedback, and charge for additional features.

Social Media Systems: Tools like Buffer, Hootsuite, and Later allow users to schedule posts across platforms but are subscription-based and impose limitations on the number of posts/accounts, making them less viable for smaller organizations or individuals.

7. Proposed System

The **Automated Email and Social Media Posting System** addresses these challenges by offering a unified, self-hosted solution for bulk communication across both email and major social media platforms.

1. Email Automation
2. Social media Automation
3. Security: SMTP login, OAuth token
4. Frontend: HTML, CSS, JAVA SCRIPT

Email Module:

1. Upload .txt file containing email addresses.
2. Input subject, message, and attachments.
3. Validate emails.
4. Authenticate via user SMTP credentials.
5. Send emails individually.

Display real-time progress/status.

Social Media Module:

1. Authenticate user via OAuth to each platform.
2. Compose post (text, image, description).
3. Choose whether to post immediately or schedule.
4. Post using platform-specific API endpoints.

8. Use Case Diagram

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. those use cases

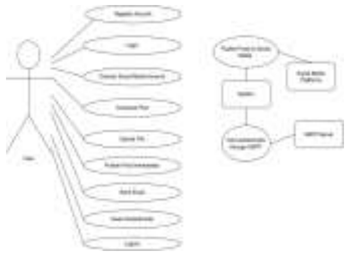


Figure: Use Case Diagram

Class Diagram

Classes:

User: user id, name, email, password

Register (), Login (), Logout

Social media account: Account id, Platform name Access Token

Connect account () disconnect account

Post: Post id, content, media URL, schedule time, status

Admin: Mange users (), manage post ()

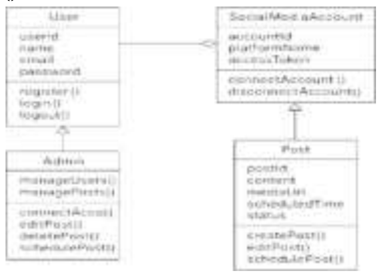


Figure: Class Diagram

Sequence Diagram*:*

1. Login
2. Confirm login success
3. Create a new post
4. Connect social media account (OAuth)
5. Post content

6. Conformation
7. Show success messages

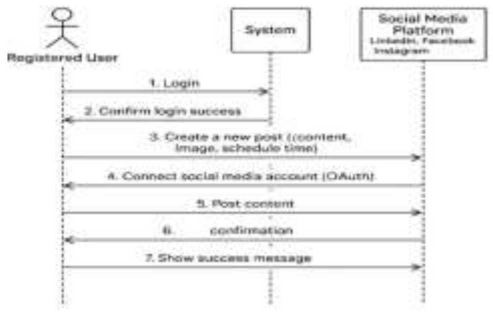
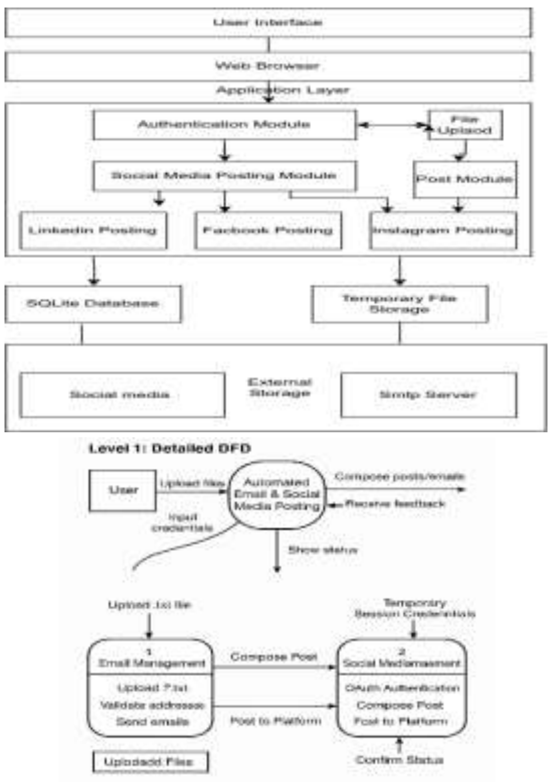


Fig: Sequence diagram

Data Flow Diagrams (DFDs)

- External Entities: User
- System: Automated Email & Social Media Posting
- Data Flows: Upload files, Input credentials, Compose posts/emails, Receive feedback
- Process 1: **Email Management**
- Upload .txt file → Validate addresses → Send emails → Show status.
- Process 2: **Social Media Management**
- OAuth Authentication → Compose Post → Post to Platform → Confirm Status. Data Stores: Temporary session credentials, Uploaded files.



Figures: Data flow Architecture of the Automated social-media and multi email posting

Activity Diagram:

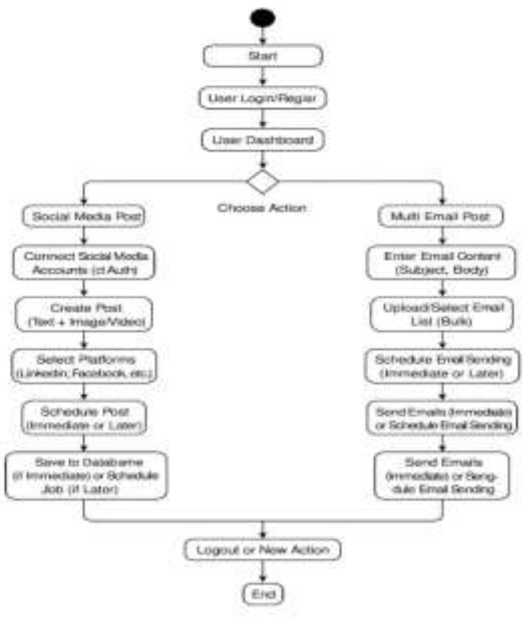

Start
User Login/Register
User Dashboard
Choose Action
Social Media Post
Connect Social Media Accounts (cl Auth)
Create Post (Text + Image/Video)
Select Platforms (LinkedIn, Facebook, etc.)
Schedule Post (Immediate or Later)
Save to Database (if Immediate) or Schedule Job (if Later)
Multi Email Post
Enter Email Content (Subject, Body)
Upload/Select Email List (Bulk)
Schedule Email Sending (Immediate or Later)
Send Emails (Immediate or Schedule Email Sending
Send Emails (Immediate or Scheg-dule Email Sending
Logout or New Action
End

Fig: Activity diagram

9. IMPLEMENTATION

Registration & Login System Registration & Account Activation

○ **User Registration Form:**
▪ Collect user details: Email, Password, Full name, Phone number, etc.
▪ Store these details in the users table with an initial is active status set to FALSE.

○ **Admin Activation:**
▪ Admins can view a list of pending users and approve or reject registrations.
▪ Once approved, the system:
• Updates is active to TRUE.
• Sets activated at with the current timestamp.
• Sends an email with an activation link.

○ **Account Activation Email:**
▪ The email contains a link (e.g., http://example.com/activate/{activation_token}) to activate the account.
▪ On activation, users are directed to a success page and can proceed to login.

Login Process

• User Login:
○ User provides email and password.
○ System checks if:
▪ The email exists in the database.
▪ The account is active (is active = TRUE).
▪ Password is correct (compare hash values).

• Session Management:
○ Upon successful login, generate a session or JWT token for authentication.
○ Optionally, log login metadata (timestamp, IP address) in login logs table for auditing.

Email Campaign Management

- *Step 1: Select Email List*
  - File Upload:
    - User uploads a .txt file containing email addresses (one per line).
    - Validate the email addresses:
      - Check for duplicate emails.
      - Validate the email format (regular expression).
      - Skip empty or malformed lines.
  - Database Storage:
    - Store valid email addresses in a recipients table with a status indicating whether it was successfully sent or skipped.
- *Step 2: Compose Message*
- 1.Email Composition:
- User enters the email subject and body.
- Support both plain text and HTML formats.
- Optional attachments: Files are uploaded and linked to the campaign.
- *Step 3: Send Emails Individually*
- Email Sending:
  - Emails are sent individually to each recipient.
  - Ensure each email is personalized and sent one-by-one to avoid spam or BCC leakage.
- Batch Process:
  - The system sends emails in batches to optimize delivery, e.g., 50 emails per batch.
- Tracking and Logging:
  - Log delivery results (success, failure) for each recipient in a delivery logs table.

- *Automated Social Media Posting*
- To enable automated social media posting, you can integrate APIs for popular platforms like Facebook, Twitter, or Instagram. Here’s how:
  - Post Composition:
    - Users create a post (text, images, links).
    - Optionally schedule the post for a future date/time.
  - Integration with Social Media APIs:
    - Use third-party libraries or APIs to post on social media (e.g., Twitter API, Facebook Graph API).
    - Ensure the system authenticates using OAuth tokens for each platform.
  - Scheduling:
    - Allow users to schedule posts for a later time.
    - Use Cron jobs or background workers to handle scheduling and publishing.
  - Logging:
    - Optionally log the post details, time of posting, and status (success/failure) for each platform in a social media posts table.

- *Automated Multi-Email Posting*

- To implement automated multi-email posting, you can adapt the existing campaign system to handle multiple campaigns simultaneously.
- Multiple Campaigns:
  - Allow users to select multiple email lists and compose different messages for each list.
  - System should send emails in parallel, processing each list individually while maintaining
  - proper logs.
- Campaign Scheduler:
  - Users can schedule different email campaigns to be sent at different times.
  - Cron jobs or background workers can be used to automate this process.
- Email Batch Management:
  - Support multiple email templates.
  - Track each campaign separately, ensuring there’s no overlap or duplication of recipient

10. OUT PUT & SCREEN SHOTS

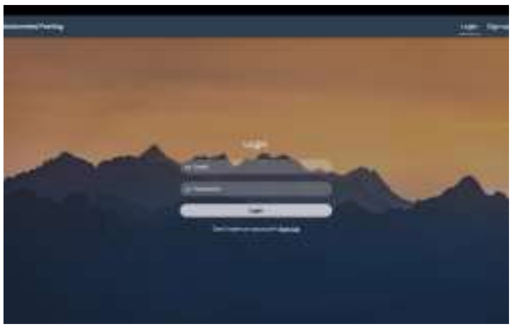


Fig: Login page

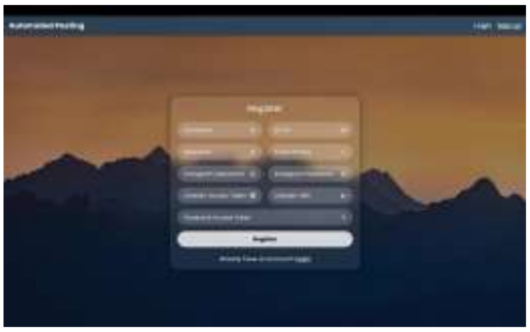


Fig: Register page

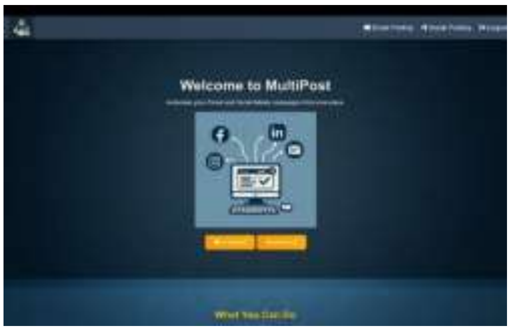


Fig :Interface page

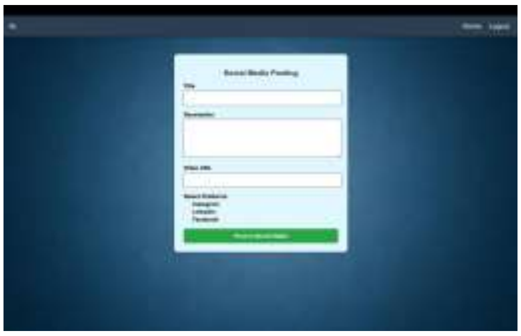


Fig: social media account

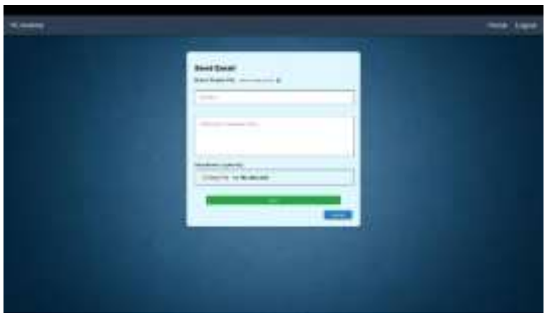


Fig Email posting

Fig: welcome page

11. CONCLUSION

In today's fast-paced digital era, timely and efficient communication plays a crucial role in both professional and personal environments. With the increasing need for mass communication, especially in corporate, academic, and marketing sectors, automation has become essential for enhancing productivity and minimizing manual effort. To address these needs, two independent systems were developed: the Automated Email Posting System and the Automated Social-Media Posting System, each focusing on specific communication channels. The Automated Email Posting System simplifies the traditional process of sending emails by allowing users to register and log in securely, upload recipient email lists via text files, compose personalized messages with attachments, and dispatch emails with real-time feedback. Meanwhile, the Automated Social-Media Posting System streamlines the scheduling and publishing of posts across LinkedIn (using URN and Access Token), Facebook (using Page ID and Access Token), and Instagram (using Username and Password). It also uses Flask for backend operations, integrating secure API-based authentication to protect user credentials while allowing users to manage posts efficiently through an intuitive web interface. Both systems maintain clean, responsive designs and emphasize strong security practices like transient credential handling and organized file management. Together, the Automated Email Posting System and the Automated social media Posting System offer a comprehensive, scalable, and robust solution for modern digital communication, making them valuable tools for organizations and individuals aiming to enhance outreach through separate, specialized platforms.

12. Reference

Email Automation & SMTP Handling:

1. Python Software Foundation. (2024). *Smtp lib — SMTP protocol client*. Retrieved from https://docs.python.org/3/library/smtplib.html
2. Real Python Team. (2024). *Sending Emails with Python*. Retrieved from https://realpython.com/python-send-email/
3. D'Souza, R. (2024). *Automating Emails Using Python and Gmail*. Towards Data Science. Retrieved from https://towardsdatascience.com/automate-email-sending-with-python-using- gmail-api-8421c8dba6f3
4. Geeks for Geeks. (2024). *Send Mail Using Flask*. Retrieved from https://www.geeksforgeeks.org/send-mail-using-flask/

Web Development (Flask, HTML, CSS, JS):

5. Flask Documentation Team. (2025). *Flask Documentation*. Retrieved from https://flask.palletsprojects.com/
6. W3Schools. (2025). *Flask Tutorial*. Retrieved from https://www.w3schools.com/python/python_flask.asp
7. Mozilla Developer Network (MDN). (2025). *HTML Documentation*. Retrieved from https://developer.mozilla.org/En-US/docs/Web/HTML
8. Mozilla Developer Network (MDN). (2025). *CSS Documentation*. Retrieved from https://developer.mozilla.org/En-US/docs/Web/CSS
9. Mozilla Developer Network (MDN). (2025). *JavaScript Guide*. Retrieved from https://developer.mozilla.org/En-US/docs/Web/JavaScript/Guide

File Handling & Attachments:

10. Flask Documentation Team. (2025). *File Upload Patterns in Flask*. Retrieved from https://flask.palletsprojects.com/En/2.2.x/patterns/file uploads/
11. Geeks for Geeks. (2024). *File Uploading in Python Flask*. Retrieved from https://www.geeksforgeeks.org/file-uploading-in-python-flask/

Security & Session Management:

1. Bhandari, A., & Singh, D. (2019). *Secured Login Using OTP and Captcha*. ResearchGate. Retrieved from https://www.researchgate.net/publication/335943545_Secured_Login_Using_OTP_and_Captcha
2. Flask Documentation Team. (2025). *Session Management in Flask*. Retrieved from https://flask.palletsprojects.com/en/2.2.x/quickstart/#sessions

3. OWASP Foundation. (2025). *Web Application Security Best Practices*. Retrieved from https://owasp.org/www-project-top-ten/

Web Technologies & System Performance:

4. Flask Framework Team. (2025). *Flask Documentation (Latest)*. Retrieved from https://

flask.palletsprojects.com/en/latest/

5. Flask-Mail Team. (2025). *Flask-Mail Documentation*. Retrieved from https://flask-mail.readthedocs.io/en/latest/
6. Mozilla Developer Network (MDN). (2025). *Front-End Web Development (HTML, CSS, JavaScript)*. Retrieved from https://developer.mozilla.org/
7. Bootstrap Team. (2025). *Bootstrap Documentation*. Retrieved from https://getbootstrap.com/ docs/
8. SQLite Consortium. (2025). *SQLite Official Documentation*. Retrieved from https:// sqlite.org/docs.html
9. Python Software Foundation. (2024). *sqlite3 Module Documentation*. Retrieved from https:// docs.python.org/3/library/sqlite3.html

Fault-Tolerant Systems (Optional Advanced Reference):

10. Lee, H., & Park, S. (2021). *Integrated Fault Reduction Scheduling (IFRS) for Reliable Systems*. IEEE Xplore. Retrieved from https://ieeexplore.ieee.org/document/9631096

Additional Email Automation Tutorials:

11. Malek, P., & Djuric, I. (2024). *Flask Send Email: Tutorial with Code Snippets*. Mail trap Blog.
12. Gigous, B. (2024). *How to Send Emails with Python + Flask-Mail*. Coding Nomads.
13. Mishra, J. (2022). *Email-Automation-using-Flask*. GitHub Repository. Retrieved from https://github.com/
14. Odyntsov, Y. (2024). *Flask Send Email Gmail: Tutorial with Code Snippets*. Mail trap Blog.
15. Tennyson, M. (2021). *How to Send Emails While Using Python Flask Fully Asynchronously*. DEV Community.

Social Media Automation (LinkedIn, Facebook APIs):

16. LinkedIn Corporation. (2025). *LinkedIn Marketing Developer Documentation*. Retrieved from https://learn.microsoft.com/en-us/linkedin/marketing/
17. Facebook Developers. (2025). *Graph API Documentation*. Retrieved from https:// developers.facebook.com/docs/graph-api/
18. Instagram Graph API Team. (2025). *Instagram Graph API Documentation*. Retrieved from https://developers.facebook.com/docs/instagram-api
19. Postman. (2025). *Using APIs for Automation*. Retrieved from https://learning.postman.com/ docs/getting-started/introduction/
20. Geeks for Geeks. (2024). *OAuth 2.0 Authorization Explained Simply*. Retrieved from https:// www.geeksforgeeks.org/oauth-2-0-authorization-code-flow/