

**CS3103 Assignment 4 Report**  
**[Option 1] Smart Mailer Program**  
**A4-ProjectGroup 30**

Wu Xingyu (A0252533L)  
Chandan Ananda Babu (A0255591X)  
Yee Jun Hyeok Bryan (A0252218L)  
Muhammad Arshad Shaik (A0239912Y)

National University of Singapore  
CS3103: Computer Networks Practice  
28th October 2024

<b>1. Introduction</b>	<b>2</b>
<b>2. Application Architecture</b>	<b>3</b>
2.1 Overview:	4
2.1.1 Client Interface:	4
2.1.2 Logging in	4
2.1.3 Status tracking	5
2.2 Email Sending Process:	6
2.3 Queue/Batching Mechanism:	7
<b>3. Features</b>	<b>8</b>
3.1 Core Features:	8
3.2 Additional Features	9
<b>4. Design Choices</b>	<b>9</b>
4.1 Technology Stack:	9
4.2 Database Choices:	10
4.3 Implementing OAuth2 with the Option to Use SMTP:	10
4.3 Email Sending Strategy:	11
4.4 Error Handling and Logging:	11
<b>5. Acknowledgments</b>	<b>11</b>
5.1 Open-source Libraries and Tools:	11
5.2 Code Generation Tools:	12
5.3 Contributions:	12
<b>Appendix</b>	<b>12</b>
Setup instructions (readme.md):	12

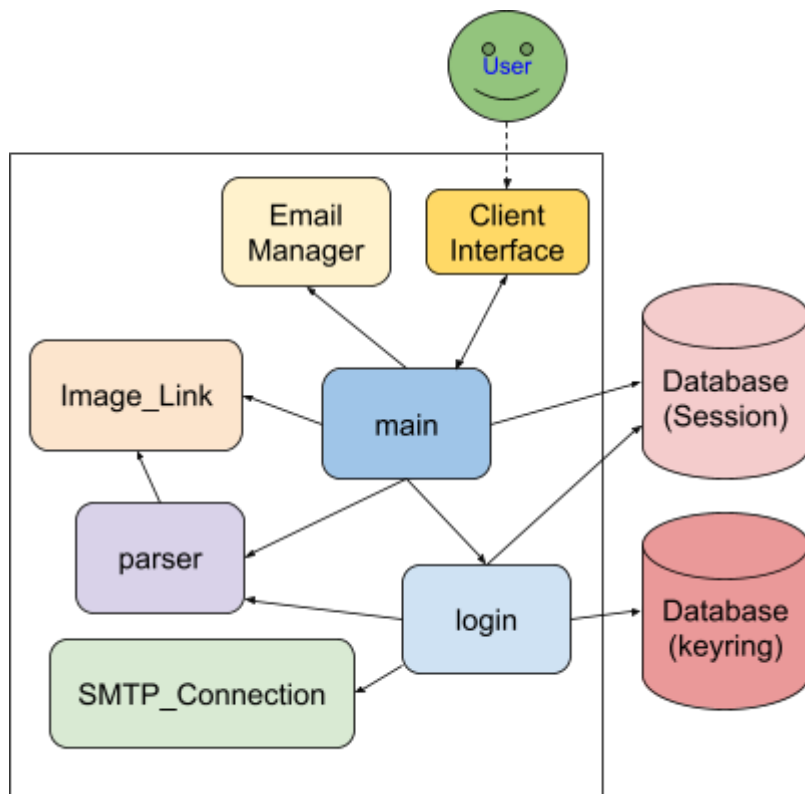
## 1. Introduction

This application allows users to easily send personalised emails in bulk and track how many times each email has been viewed. The view count can also be seen at a later time, even after the web app has been closed.

Instructions to download and set up the program can be found in the appendix. A more user-friendly guide can be found in the GitHub repository:

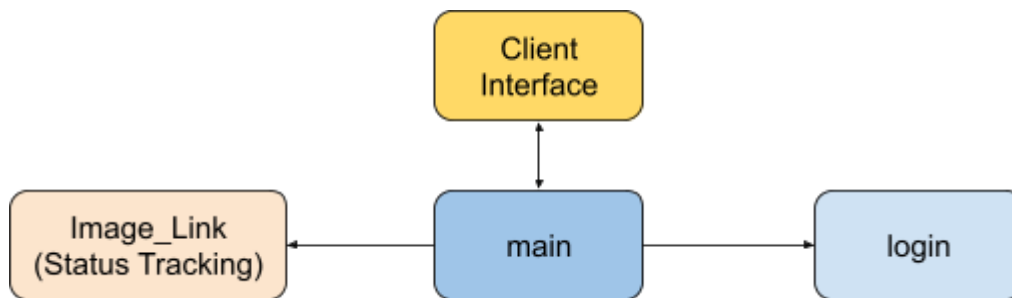
<https://github.com/Chandan8186/cs3103-grp-30/blob/main/README.md>

## 2. Application Architecture



The architecture diagram above shows the main components of the Smart Mailer and how each component interacts with one another. Note that 'Client Interface' is not an actual module, and represents the user-facing Web GUI ran with Flask.

## 2.1 Overview:



### 2.1.1 Client Interface:

The UI for this application is a web application hosted on localhost to communicate with the email server. The mail data .csv file (for recipient list) and the mail body .txt file (subject and body of email) are uploaded to the web application. An optional department code can be filled in, to only send emails to recipients in that department (leaving it blank / filling in “all” will send to all departments). There is a preview page for users to check if the emails are correct, and a submit button to actually send it.

### 2.1.2 Logging in

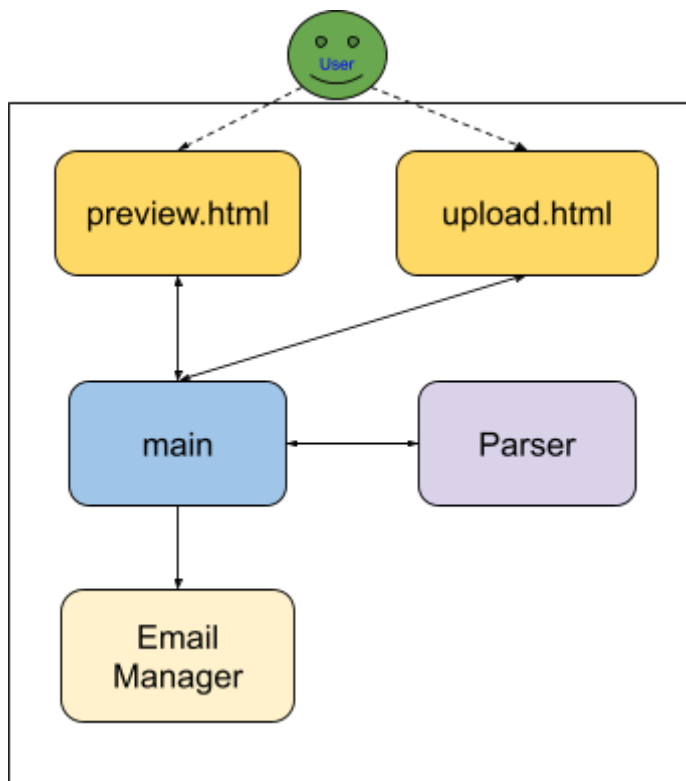
Saving, loading, and managing login states is implemented using a combination of keyring, flask-login, and flask-dance. Most details are stored within session cookies, which are signed with a secret key to ensure they are not tampered with. They are however not encrypted, so sensitive information such as passwords are instead stored using the python keyring module. Flask-login is used to automatically save unique ids as session cookies, which can then be used by keyring to ensure the unique id matches with our keyring database within the computer’s credential manager.

For SMTP users, passwords are also securely stored in the keyring, allowing for persistence of email and passwords. Validity of email and password pairs are however not checked and left to the SMTP server to verify. For OAuth users, no passwords are stored in the keyring. Flask-dance is used instead to manage logging in, authorising specified scopes, and storing & retrieving OAuth tokens from the session cookie. However, OAuth users need to create their authentication applications via Google cloud (for Gmail users) or Microsoft Azure (for Outlook users) to specify the necessary scopes, redirect URIs, etc. for authentication using flask-dance.

### 2.1.3 Status tracking

Status tracking is implemented by generating a redirect link to a 1x1 transparent image, by using a 3rd party API (ulvis.net) that allows for tracking of visits to the redirect link. Since most email clients would automatically download and display embedded images, the redirect link would automatically be used when the user opens the email, allowing for effective tracking of view counts. Moreover, since a 3rd party server is used for the redirects, we are still able to keep track of the view counts even after our web app is closed, allowing for accurate numbers to be seen. As the server requests to Ulvis are slow, requests are made asynchronously. It is rate limited at 25 requests at a time so that we would not abuse their free server. An expiry date has also been set for the redirect links to prevent too many redirect links from being created. The expiry date is set to 90 days, which would likely be more than sufficient for the purpose of this application.

## 2.2 Email Sending Process:



An **Email\_Manager** object is first created to manage the batch sending and rate limiting of emails.

Assuming the user has already logged in, they can upload the mail data CSV file containing the email address, name and department code (plus up to 3 additional optional fields) of each recipient, as well as the mail body text file, containing the subject and body (latter in HTML format). They may add a department code filter or leave it blank, in which case all recipients will receive the email.

Once the user clicks **Preview before sending**, the `parser.py` file will parse the data in the uploaded files and store it in a `Parser` object. It will then use the data and department code filter to produce a list of emails, each email being a dictionary containing subject (str), body (str) etc. The user is then directed to a preview page showing the subject and body of the first recipient's email and the placeholders (e.g. `#name#`).

Once the user clicks **Send Emails**, `email_manager` will create one thread to start sending emails in batches. Meanwhile, the user is brought to the summary page. The summary page will show a live update of whether each email is sent successfully, along with their view count. While an email is being sent out, the user will not be allowed to send another batch of emails.

### 2.3 Queue/Batching Mechanism:

A single new thread is created to send the emails one by one at a maximum of 20 emails every minute (62s in code for redundancy purposes). This number was chosen as certain Outlook clients are limited to only 20 emails per minute. A new thread is used to allow users to view the results page before all the emails are sent, while still allowing user interaction and asynchronous image count retrieval to run on the main thread. A status column also indicates whether a particular email has been sent. This implementation moreover allows the emails to continue sending even after the user changes their view within the web app, or even after they close their tab. This is useful if the user is sending hundreds of emails at once, while misclicking into another view or refreshing the page.

The user is prevented from sending another batch of emails until the current one has been fully sent, preventing a bypass of the rate limit. Furthermore, refreshing the page (or going to the previous page and back within their browser) does not resend the already sent emails. The user has to reselect the CSV and body files and go through the whole sending process again in order to send another batch, preventing accidental duplicate sending of emails.

### 3. Features

#### 3.1 Core Features:

- Preview emails before sending.
  - Before sending the emails out, the user is brought to a preview page showing a sample preview of the email to be sent to the first recipient. This allows the user to easily do one final check before actually sending the emails.
- Support customizable email subject and body
  - The user must include 'email', 'name', and 'department' fields, and may choose to include up to 3 additional unique fields.
  - The user may add placeholders (e.g. #name# and #department#) within the template subject and body and they will be replaced with the relevant recipient's details when the email is sent.
- Send emails in batches with rate limiting (rate limit of 20)
  - Emails are sent using a separate thread with a maximum rate limit of 20 emails per minute to reduce the likelihood of throttling and blacklisting by email providers.
  - The email summary page updates live on whether each email is successfully sent out.
- View counters to check if each user has clicked on the link for their email
  - The upload summary page shows a table of all emails that were sent out, including whether each email was sent successfully and a counter denoting how many times each email was opened by the relevant recipient.



### 3.2 Additional Features

- Support login with OAuth for both Google and Outlook has been added, as they are most commonly used email providers. This is useful since Google requires making app passwords for login, and Outlook has deprecated basic auth, providing users another method of logging in.
- Allow cancelling of emails that are still being sent. This is useful since emails are sent in batches of 20 per minute, allowing time for users to change their mind and cancel sending the remaining emails.
- The user can retrospectively check the view counter of any previously sent email by uploading the relevant email files (mail data csv and mail body txt), typing the filtered department and clicking **View counts of specified CSV and Body**.
- The user can retrospectively check the 'sent' and 'view count' status of the most recent batch of emails by clicking **Check most recently sent emails**.  
This works even after changing to a different screen, or checking view counts of other CSV and body emails.

## **4. Design Choices**

### 4.1 Technology Stack:

#### 4.1.1 Languages, frameworks, and libraries used for *Python* web application:

- HTML5, CSS3: Frontend webpage
- jQuery: Update html elements periodically, without refreshing the page
- Python3: Backend
- flask-login: Manage user credentials
- WTForms: Obtain details securely from user using CSRF

- smtplib: Establish SMTP connection with SMTP server
- flask-dance: Establish and maintain OAuth connections
- aiohttp: Asynchronously send HTTP requests for image counts

#### 4.2 Database Choices:

- Maintain less-important details with session cookies for ease of use
- Stores credential information such as passwords with python keyring module, storing sensitive information in the computer's in-built credential manager

#### 4.3 Implementing OAuth2 with the Option to Use SMTP:

It was observed that carrying out the SMTP procedure shown in CS3103 Lecture 4 Slide 39 would not work when connecting to Outlook's SMTP server. This is because basic authentication was recently disabled. Microsoft instead advocates for the use of OAuth2 mechanism to authenticate and connect to Outlook's mail server.

[<https://learn.microsoft.com/en-us/exchange/clients-and-mobile-in-exchange-online/deprecation-of-basic-authentication-exchange-online>]

Currently, Gmail users are still able to connect to SMTP servers directly through the basic SMTP procedure. However, this would change soon since Gmail is also deprecating basic authentication using SMTP and shifting to OAuth.

[<https://support.google.com/a/answer/14114704?hl=en>]

As such, we have opted to implement OAuth2 for Gmail and Outlook users in anticipation of these changes.

#### 4.3 Email Sending Strategy:

A separate thread is used to send the emails, so that users are still able to interact with the website as it is sending. The emails will continue sending in the background even if the user changes view by refreshing the page or changing to the home page. This allows for uninterrupted sending of emails, which should be the main focus of the app. Since we are rate limited by the email server, there is no point in using async or using multithreading to send many emails at once, so each email in a batch is sent synchronously, before waiting for the 1 minute to be up. This further reduces the chances of being flagged by the email provider.

The delay between each batch was also decided based on observation of certain Outlook clients only permitting 20 emails per minute. This was the most aggressive rate limit among all the email providers we have tested (Gmail, Outlook).

#### 4.4 Error Handling and Logging:

Most errors are caught and displayed on the web page itself using flask.flash. More technical and detailed errors are printed on the console for the developer to see.

### **5. Acknowledgments**

#### 5.1 Open-source Libraries and Tools:

- flask
- flask\_dance
- flask\_login
- oauthlib
- wtforms
- aiohttp

- pandas
- keyring
- ulvis.net
- jQuery

## 5.2 Code Generation Tools:

OpenAI's GPT for boilerplate code for reference

OpenAI's GPT for explanation and tools suggestion

## 5.3 Contributions:

1. Steps to create and register an authentication application to perform OAuth2 login for Outlook Users

<https://learn.microsoft.com/en-us/entra/identity-platform/scenario-web-app-sign-user-app-registration?tabs=python>

2. <https://ulvis.net> for redirect url APIs and status viewing

3. wikipedia.org & wikimedia.org for the 1x1 transparent image:

<https://upload.wikimedia.org/wikipedia/commons/c/ca/1x1.png>

## **Appendix**

Setup instructions (readme.md):

# Appendix A: Set-up and Usage Instructions

---

This is extracted from the [README](#) of the source code repository. For troubleshooting, please refer to the [README](#).

## Contents

- [Setting up](#)
- [Features](#)
  - [Logging in \(OAuth\)](#)
  - [Logging in \(Password\)](#)
  - [Uploading files](#)
  - [Previewing Email](#)
  - [View counts](#)
  - [Logging Out](#)

## Setting up Smart Mailer Program

### Step 1:

Navigate to `cs3103-grp-30` folder (which contains `main.py`) located inside the submission ZIP. Alternatively, git clone the repository and navigate to the directory containing `main.py`:

```
git clone https://github.com/Chandan8186/cs3103-grp-30.git
cd cs3103-grp-30
```

### Step 2:

Install the required packages with pip:

```
pip install -r requirements.txt
```

### Step 3:

Launch the program with the following command:

```
# (on Windows)
python main.py

# (on MacOS or Linux)
python3 main.py
```

A successful launch on the localhost server should show the following:

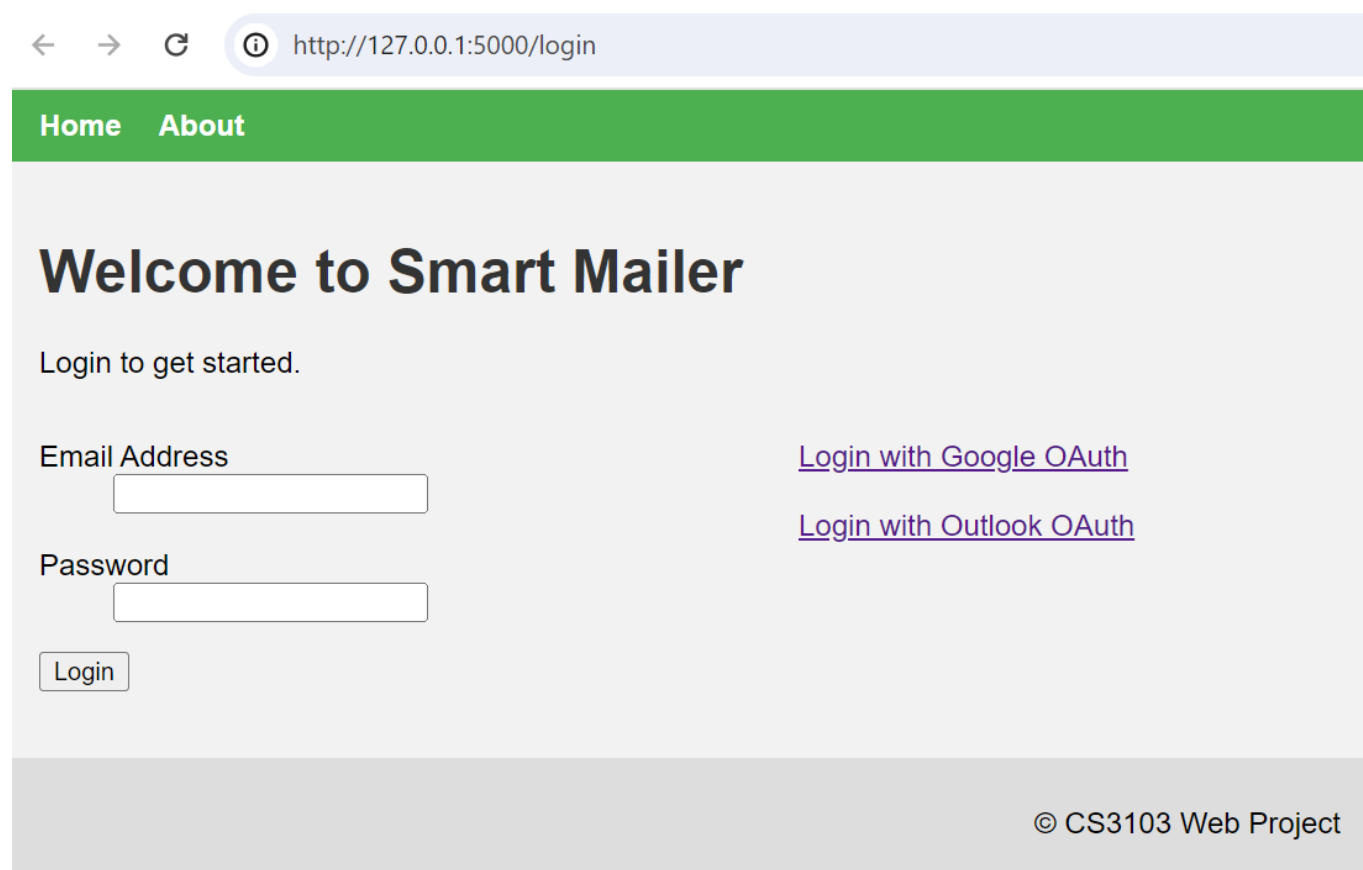
```
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 120-854-820
127.0.0.1 - - [04/Nov/2024 18:13:00] "GET / HTTP/1.1" 302 -
127.0.0.1 - - [04/Nov/2024 18:13:00] "GET /login HTTP/1.1" 200 -
127.0.0.1 - - [04/Nov/2024 18:13:00] "GET /static/style.css HTTP/1.1" 200
-
127.0.0.1 - - [04/Nov/2024 18:13:01] "GET /favicon.ico HTTP/1.1" 404 -
```

#### Step 4:

Open the application on a web browser using the following URL:

- <http://localhost:5000>, if you are planning to use Outlook OAuth login for the app, or
- <http://127.0.0.1:5000> otherwise.

Opening the web application should lead to the following page:



The screenshot shows a web browser window with the address bar displaying <http://127.0.0.1:5000/login>. The page has a green header bar with "Home" and "About" links. The main content area is light gray and features the heading "Welcome to Smart Mailer" in a large, bold, dark font. Below the heading, it says "Login to get started." There are two input fields: "Email Address" and "Password", each with a corresponding text label to its left. To the right of the "Email Address" field, there are two links: "Login with Google OAuth" and "Login with Outlook OAuth", both in purple text. Below the "Password" field is a "Login" button. At the bottom right of the page, there is a footer that reads "© CS3103 Web Project".

## Features

The smart mailer program allows you to schedule multiple batches of emails at once, and track their send status and view counts with ease.

### Logging into your account using OAuth

Google and Outlook OAuth can be used for this web application.

Generating a client id and client secret is necessary to use OAuth (if you are hosting this application yourself). You may follow the [Google](#) or [Outlook](#) official documentations to do so. Do remember to include the following authorized URIs:

```
(Outlook)
http://localhost:5000/login/azure/authorized

(Google)
http://127.0.0.1:5000/login/google/authorized
```

Here is an example:

## Authorized JavaScript origins

For use with requests from a browser

URIs 1 \*  
http://127.0.0.1:5000

[+ ADD URI](#)

## Authorized redirect URIs

For use with requests from a web server

URIs 1 \*  
http://127.0.0.1:5000/login/google/authorized

[+ ADD URI](#)

Paste the client ID and secret into `main.py` as required. You may need to relaunch the web application.

```
google_bp = make_google_blueprint(  
    client_id="",  
    client_secret="",  
    redirect_to="login_google",  
    offline=True,  
    scope=["https://www.googleapis.com/auth/userinfo.email", "https://www.goo  
)  
azure_bp = make_azure_blueprint(  
    client_id="",  
    client_secret="",  
    redirect_to="login_azure",  
    scope=["https://graph.microsoft.com/User.Read", "https://graph.microsoft.  
)
```

## Logging into your SMTP account

Fill in the login details for your email account. Do note that some email providers such as Gmail requires users to generate and use an application password instead of their actual password. You may follow the link [here](#) to set up your app password.

**Note: Outlook has deprecated basic auth and SMTP may not work properly. Please use OAuth Authentication instead. You may refer to the following instructions for [logging into your account using OAuth](#).**

## Uploading files

After successfully logging in, upload the CSV file containing recipient details and the mail body file containing the email body, and enter the department code you wish to send to.

Please ensure the files are in the appropriate format. You may view the provided sample maildata.csv and body.txt files as an example.



# Upload your input files.

**Mail data .csv file format\*:**

email	name	department
...	...	...

\* You may add up to 3 more (unique) fields in addition to name and department.

**Mail body .txt file format\*\*:**

Line 1: subject (can be empty)

Line 2: empty line

Line 3 onwards: non-empty body (in HTML format)

\*\* The placeholders are #FIELD\_NAME# e.g. #name#, #department#

Upload Mail Data CSV: 

Choose File

 No file chosen

Upload Mail Body Text File: 

Choose File

 No file chosen

Enter department code (type "all" or leave empty to send to all departments):

Preview before sending

View counts of specified CSV and Body

Check most recently sent emails

Preview and Sending Emails

After selecting your files, you may click on "Preview before sending" to verify whether the emails are correct. The first email will be shown with its placeholders replaced.

Please ensure the emails are correct before sending!

Department to send to: All Departments

Placeholders\*: #name#, #department#

First Recipient's Preview:

Subject: Welcome to the HR Team, Alice Smith!

Body:

Dear Alice Smith,

We are **excited** to welcome you to the HR department at our company. Your expertise and enthusiasm will be a valuable addition to our team.

Please let us know if you have any questions!

Best regards,

John

HR Manager, ABC Company

\* Note that the placeholders will be replaced with the relevant recipient's field respectively.

Are you sure you want to send the emails?

Return to file upload page

Send Emails

View counts

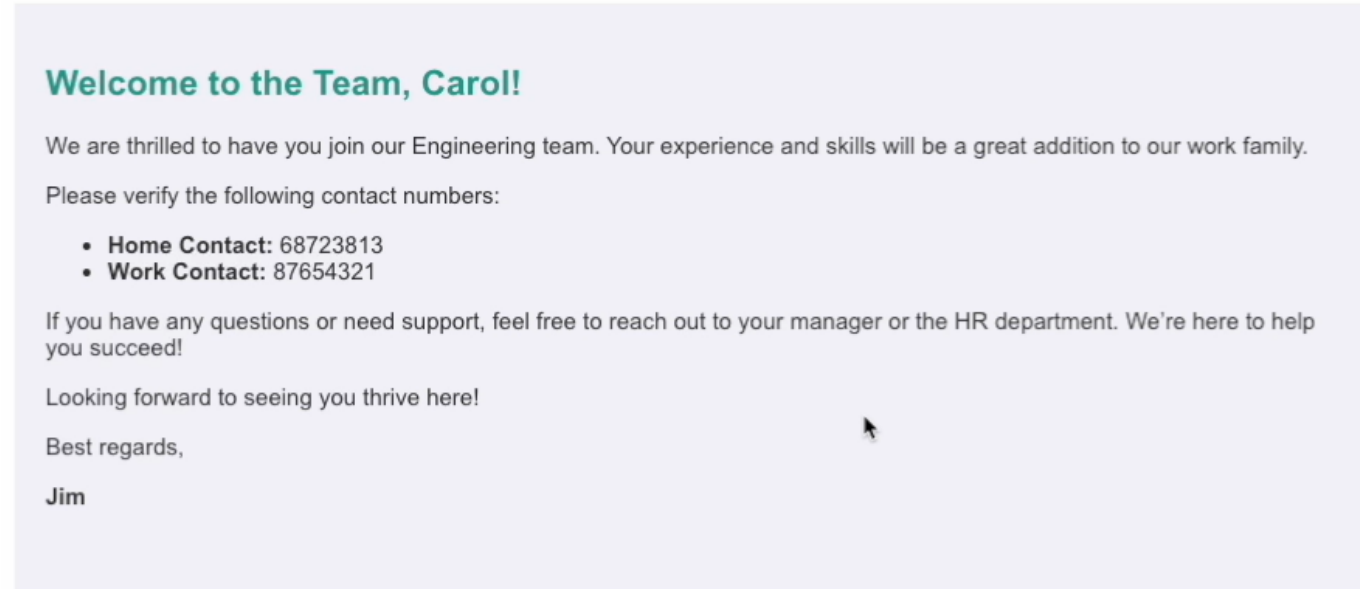
If the mail is sent successfully, the following information would be displayed on the screen, showing the send status and the view count:

HomeAbout

Logged in as chandanandababu@gmail.comLogout

Processed Emails							Cancel Sending Emails	
Email	name	department	home	work	Subject	Body	Send Status	View Count
chandan_ab@outlook.com	Alice	HR	67862378	84839721	Welcome to the Team, Alice! We're Excited to Have You Onboard!	<div>Welcome to the Team, Alice!</div> <div>We are thrilled to have you join our HR team. Your experience and skills will be a great addition to our work family.</div> <div>Please verify the following contact numbers:</div> <div><div>• Home Contact: 67862378</div><div>• Work Contact: 84839721</div></div> <div>If you have any questions or need support, feel free to reach out to your manager or the HR department. We're here to help you succeed!</div> <div>Looking forward to seeing you thrive here!</div> <div>Best regards,</div> <div>Jim</div>	✓	0
						<div>Welcome to the Team, Bob!</div> <div>We are thrilled to have you join our HR team. Your experience and skills will be a great addition to our work family.</div> <div>Please verify the following contact numbers:</div>		

When a user opens the mail, the inbuilt redirect link tracker would trigger:



The view count would increase as follows:

						Jim	
chandan_ab@outlook.com	Carol	Engineering	68723813	87654321	Welcome to the Team, Carol! We're Excited to Have You Onboard!	<p><b>Welcome to the Team, Carol!</b></p> <p>We are thrilled to have you join our Engineering team. Your experience and skills will be a great addition to our work family.</p> <p>Please verify the following contact numbers:</p> <ul style="list-style-type: none"><li>• <b>Home Contact:</b> 68723813</li><li>• <b>Work Contact:</b> 87654321</li></ul> <p>If you have any questions or need support, feel free to reach out to your manager or the HR department. We're here to help you succeed!</p> <p>Looking forward to seeing you thrive here!</p> <p>Best regards,</p> <p>Jim</p>	1

Logging out

You may press the logout button at the top right corner of the webpage to logout.