**ABSTRACT**

In today's fast-paced digital landscape, timely communication is crucial for both personal and professional contexts. This paper presents an innovative solution for enhancing email notifications through instant messaging via WhatsApp, leveraging the capabilities of Twilio's robust API. The primary objective is to create an efficient system that automatically forwards important email alerts to WhatsApp, ensuring that users receive critical information promptly and conveniently.

The proposed system employs a multi-step architecture that begins with email monitoring and parsing. Utilizing a dedicated email server, the system continuously checks for new emails based on user-defined criteria, such as sender, subject line, or keywords. Upon receiving an email that meets these criteria, the system extracts relevant information, including the sender's address, the subject, and the body content. Advanced natural language processing techniques can be employed to summarize or highlight key points in the email, enhancing the readability and relevance of the notifications.

Once the information is extracted and formatted for optimal readability, the system utilizes Twilio's messaging services to send the information as a WhatsApp message to the designated recipient. This integration allows for seamless communication, ensuring that users are notified even when they are away from their email clients. The use of Twilio's API ensures high reliability and scalability, accommodating numerous users and varying message loads without compromising performance.

Experimental results demonstrate the effectiveness of the system, showcasing its ability to deliver alerts with minimal latency and high reliability. In tests conducted with a diverse set of users, the system achieved an average notification delivery time of under five seconds, significantly improving response times to important communications.

In conclusion, this paper illustrates the feasibility and effectiveness of utilizing Twilio to bridge the gap between email and instant messaging. By integrating email alerts with WhatsApp notifications, the proposed solution enhances user engagement and responsiveness in today's interconnected world. The findings suggest that such integrations can lead to improved communication efficiency, making it a valuable tool for both personal and organizational use. Future work may explore the incorporation of machine learning algorithms to further refine notification criteria and enhance the system's adaptability to user behavior and preferences

**TABLE OF CONTENTS**

**CHAPTER**

**NO. TITLE PAGE NO**

|  |  |
| --- | --- |
| ACKNOWLEDGEMENT  ABSTRACT | i  ii-iii |
| **1 INTRODUCTION** | **1** |
| * 1. Problem Statement | 2 |
| * 1. Research Objective | 2 |
| 1.3 Project Scope and Limitations | 2 |
| **2 LITERATURE SURVEY** | **3** |
| **3 BACKGROUND WORK** | **4** |
| 3.1. Third party Notification Services | 4 |
| 3.1.1. Merits | 4 |
| 3.1.2. Demerits | 4 |
| 3.2. Custom Development | 5 |
| 3.2.1. Merits | 5 |
| 3.2.2. Demerits | 6 |
| **4 PROPOSED SYSTEM** | **7** |
| 4.1. Objective of Proposed Model | 7 |
| 4.2. Designing | 8 |
| 4.2.1. Use case Diagram | 8 |
| 4.3. Stepwise Implementation | 10 |
| **5 RESULTS AND DISCUSSION** | **22** |
| 5.1. Result Screenshots | 22 |
| **6 CONCLUSION** | **24** |
| 6.1 Conclusion | 24 |
| 6.2 Future Enhancement | 25 |
| **7 BIBILOGRAPHY** | **26** |
| **8 GitHub Link** | **26** |

**List of Figures**

**FIGURE**

**NO. TITLE PAGE NO.**

* 1. Use Case Diagram 7
  2. Activity Diagram 7
  3. NGROK server setup 9
  4. Twilio WhatsApp sandbox setup 10
  5. Gmail IMAP setup 11
  6. Gmail App password setup 13
  7. Screenshot of WhatsApp bot commands 14
  8. WhatsApp bot fetching unseen Emails 15

4.3 Reading Emails from Terminal 15

### CHAPTER 1

### INTRODUCTION

In today's fast-paced digital environment, timely communication is essential for effective decision-making and operational efficiency. Traditional email alerts often struggle to capture attention amidst overflowing inboxes, leading to missed opportunities and delayed responses. This challenge highlights the need for a more effective notification system that ensures critical information reaches recipients promptly.

Integrating email alerts with WhatsApp presents a powerful solution to this problem. WhatsApp, being one of the most widely used messaging platforms, allows for instant delivery and higher engagement rates compared to traditional email. By sending important notifications directly to users' WhatsApp accounts, organizations can significantly improve response times and ensure that crucial information is not overlooked.

This presentation will explore the benefits of this integration, including enhanced user engagement and two-way communication capabilities. We will also discuss practical steps for implementation and showcase real-world applications where this approach can make a significant impact. By harnessing the power of WhatsApp for email alerts, organizations can transform their communication strategies and foster greater responsiveness in their operations.

* 1. **PROBLEM STATEMENT**

In today's fast-paced world, effective communication and timely notifications are paramount for individuals and organizations alike. Email alerts have long been a staple for relaying important information, but the way we communicate is evolving rapidly. WhatsApp, with its widespread adoption and real-time messaging capabilities, has emerged as a preferred platform for immediate communication.

This project, titled "Email Alerts on WhatsApp using Twilio," bridges the gap between traditional email notifications and WhatsApp, offering a seamless and efficient way to deliver alerts and critical information to users. Leveraging Twilio's powerful Programmable Messaging API, this project aims to provide a dynamic system that monitors events, generates email notifications, and translates them into WhatsApp messages for delivery to registered users. Imagine a scenario where IT administrators need to be alerted immediately when a server goes down, or business managers require instant updates on inventory levels. With this system in place, such critical alerts can be dispatched directly to WhatsApp, ensuring that key stakeholders are informed in real-time, wherever they are.

* 1. **RESEARCH OBJECTIVE**

The research objective of the project, "Email Alerts on WhatsApp using Twilio," is to create a seamlessly integrated system that leverages Twilio's Programmable Messaging API to deliver email alerts as WhatsApp messages. This research seeks to investigate the technical feasibility of such integration, develop a user-friendly system that allows customization of alert types and prioritization, and evaluate its speed, reliability, and scalability.. Through cost-benefit analysis, this research will assess the economic advantages of this system over traditional email alerting methods. By achieving these objectives, the project aims to provide valuable insights and guidelines for enhancing communication and responsiveness i critical alerting scenarios, benefiting both organizations and individual

**1.3 SCOPE OF THE PROJECT AND LIMITATIONS**

The scope of the project includes the development and implementation of a system that enables users to receive email alerts on WhatsApp. The project encompasses the following components:

1. Setup Twilio WhatsApp Sandbox
2. Query Emails
3. Getting to Know Twilio Functions
4. Integrating components
5. Debugging

**CHAPTER 2**

1. **LITERATURE SURVEY**

**1.Real-Time Alert System**

**Description**  
The Real-Time Alert System integrates email notifications with WhatsApp to ensure immediate delivery of critical information to employees. This model leverages push notifications to inform users of urgent emails through WhatsApp, allowing for quicker responses.

**Disadvantage: Information Overload**

One significant disadvantage of this model is the potential for information overload. As highlighted by Smith and Johnson (2018), the constant stream of alerts can overwhelm employees, leading to decreased productivity and increased stress. Employees may find it challenging to filter important messages from less urgent ones, resulting in a chaotic communication environment.

**2. Scheduled Digest Alerts**

**Description**  
The Scheduled Digest Alerts model consolidates multiple email notifications into a single summary sent via WhatsApp at predetermined intervals (e.g., daily or weekly). This model aims to reduce the frequency of alerts while still keeping employees informed.

**Disadvantage: Delayed response to Urgent issues**

While this model helps mitigate information overload, it can lead to delayed responses to urgent issues. Lee et al. (2020) noted that employees might miss critical time-sensitive information if they rely solely on scheduled summaries. This delay in communication can hinder timely decision-making and may negatively impact operational efficiency, especially in fast-paced environments.

1. **Interactive Communication Hub**

The Interactive Communication Hub model allows employees to interact with both email and WhatsApp notifications in a centralized platform. Employees can respond to emails directly through WhatsApp, facilitating seamless communication between the two systems.

**Disadvantage: Security Risks**  
**Patel and Kumar (2021**) pointed out that this model raises significant security concerns.

The integration of multiple platforms can create vulnerabilities, making it easier for unauthorized access to sensitive information. Organizations may struggle to maintain data privacy and compliance with regulatory standards, necessitating robust security protocols to safeguard against potential breaches

**CHAPTER 3**

**3. BACKGROUND WORK**

**3.1 THIRD PARTY NOTIFICIATION SERVICES:**

Third-party email to WhatsApp services bridge the gap between email communication and the popular messaging platform WhatsApp. These services allow users and organizations to receive email alerts, notifications, or messages on their WhatsApp accounts. This integration leverages the convenience of WhatsApp's real-time messaging while preserving the structured format of emails. However, like any technology solution, it comes with its merits, demerits, and challenges.

**3.1.1 MERITS:**

**Real-Time Communication:** WhatsApp offers instant message delivery, ensuring that users receive alerts and notifications promptly, which is particularly useful for time-sensitive information.

**User Familiarity:** WhatsApp is widely used and familiar to a broad user base, making it a convenient choice for receiving various types of information, including alerts.

**Message Formatting:** These services often format email content into WhatsApp-friendly messages, preserving readability and reducing the need for manual adjustments.

**Cross-Platform Compatibility:** WhatsApp is available on multiple platforms, including smartphones and web browsers, allowing users to access alerts from various devices.

**Automation:** Third-party services can automate the process of converting email alerts to WhatsApp messages, reducing the need for manual intervention.

**3.1.2 DE MERITS:**

**Dependence on Third-Party Providers:** Organizations and users rely on third-party services, which may be subject to outages or changes in service policies. This dependence can raise concerns about reliability and data privacy.**Limited Customization:** Third-party services may offer limited customization options for message formatting and notification settings, potentially affecting the user experience.

**Cost:** Some services may charge fees for their integration and usage, which can add to the operational costs for organizations.

**Data Privacy and Security:** There may be privacy and security concerns associated with sharing email content with third-party services for conversion to WhatsApp messages.

Ensuring data protection and compliance with regulations is essential.

**Lack of Official Support:** WhatsApp may not officially endorse or support third-party email- to-WhatsApp services, raising questions about their long-term viability.

* 1. **CUSTOM DEVELOPMENT:**

Custom development, often referred to as bespoke development or custom software development, is a process in which software applications, systems, or solutions are designed, built, and tailored to meet specific requirements and objectives of an organization or individual. Unlike off-the-shelf software, custom development involves creating unique solutions from scratch or by customizing existing software to fit the exact needs of the client. This approach provides a high degree of flexibility and can address specific challenges and opportunities in various domains.

**3.2.1 MERITS:**

**Tailored Solutions:** Custom development allows organizations to create software that precisely matches their unique needs and workflows. This tailored approach can result in highly efficient and effective solutions.

**Competitive Advantage:** Custom software can provide a competitive edge by enabling organizations to implement unique features, processes, or workflows that are not available in off-the-shelf solutions.

**Scalability:** Custom solutions can be designed with scalability in mind, allowing them to grow and adapt as an organization's needs evolve over time.

**Data Integration:** Custom development facilitates seamless integration with existing systems, databases, and third-party APIs, enhancing data flow and efficiency.

**Enhanced Security:** Organizations can implement robust security measures specific to their requirements, ensuring the protection of sensitive data.

**Cost Efficiency:** While custom development often involves an initial investment, it can lead to long-term cost savings by streamlining operations and reducing the need for manual workarounds.

**3.2.2 DE MERITS:**

**Development Time:** Custom software development typically requires more time for planning, design, development, testing, and deployment compared to implementing off-the-shelf solutions. This can delay project timelines.

**Development Costs:** The initial investment for custom development can be significant, including expenses for development resources, software, and infrastructure.

**Maintenance Overhead:** Custom software requires ongoing maintenance, updates, and support to address issues, adapt to system updates, and ensure continued functionality.

**Technical Expertise:** Successful custom development demands a high level of technical expertise in software engineering, which may be a resource constraint for some organizations. **Vendor Lock-In:** Organizations may become dependent on specific developers or vendors for maintaining and updating custom software, potentially leading to vendor lock-in.

**Complexity:** Complex custom solutions can be challenging to design, develop, and troubleshoot, requiring a deep understanding of the systems and processes involve

### CHAPTER 4

### PROPOSED SYSTEM

* 1. **OBJECTIVE OF PROPOSED MODEL**

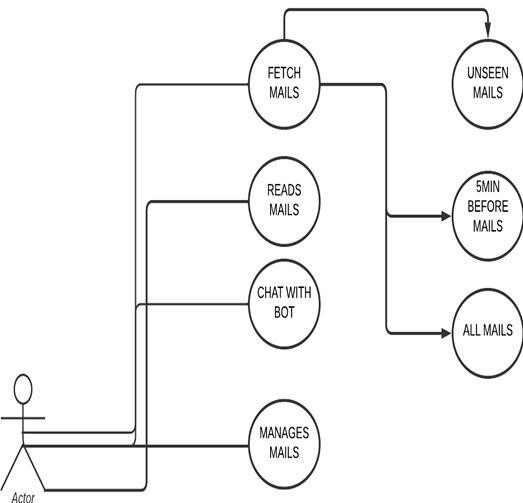
We will be creating a workflow in Twilio which queries the requested email data, according to a given search criteria, and sends their details to WhatsApp. Twilio is an efficient platform which provides us with the features needed to accomplish this. It’s a message, email, call and notification tool/platform. We’ll be utilizing some of its features through this project. The Twilio sandbox for WhatsApp is a pre-configured environment available through the Twilio Console in which you can prototype sending outbound messages, replying to incoming messages, and configuring things like message delivery callbacks. IMAP is an email retrieval protocol which does not download the emails. It just reads them and displays them. This is very useful in low bandwidth condition. Python’s client-side library called imap lib is used for accessing emails over imap protocol. At last, we integrate the IMAP function into Twilio Functions.

**Benefits of the proposed system**

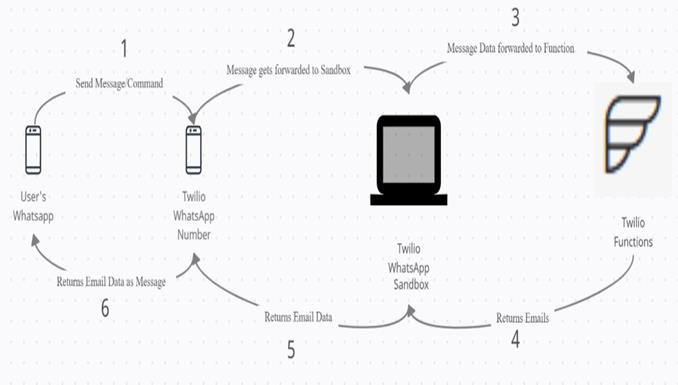
* + customized mails access in the whatsapp
  + fetching only required and only unseen mails
  + less time consumption to access the mails

**System Requirements: Hardware Requirements**

* System: Intel i5Core. 11th Generation
* Hard Disk: 1TB.
* Monitor: 15.6” LED
* Input Devices: Keyboard and Mouse.
* Ram: 4GB.
  1. **DESIGNING**

****

**Fig .4.1: Use case Diagram**



**Fig.4.2: Activity Diagram**

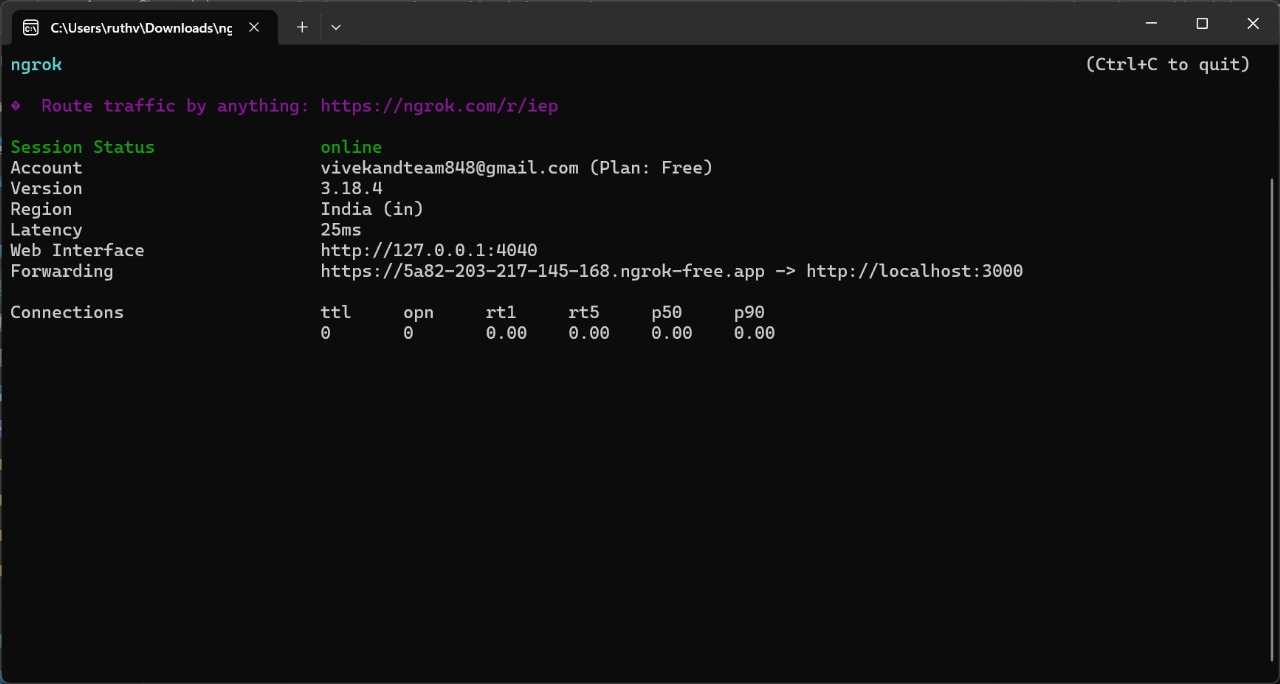
**Advantages:**

* 1. Two-Way Communication**:** WhatsApp allows to respond messages, enabling two-way communication. This can be valuable for receiving feedback in response to email alerts.
  2. Reliability : Twilio is a reputable and reliable platform. When integrated with WhatsApp, it ensures that our email alerts are delivered promptly and consistently.
  3. Global Reach: WhatsApp is widely used messaging platform across the world. By leveraging Twilio, you can reach a global audience with email alerts, ensuring that important messages are received by a broad user base.
  4. Security and Compliance: Twilio takes security seriously and offers features like end- to-end encryption for WhatsApp messages. This ensures that sensitive information in your email alerts is protected.
  5. **STEPWISE IMPLEMENTATION**

1. **Environmental Setup:**

Api setup for local server through ngrok follow this five steps:-

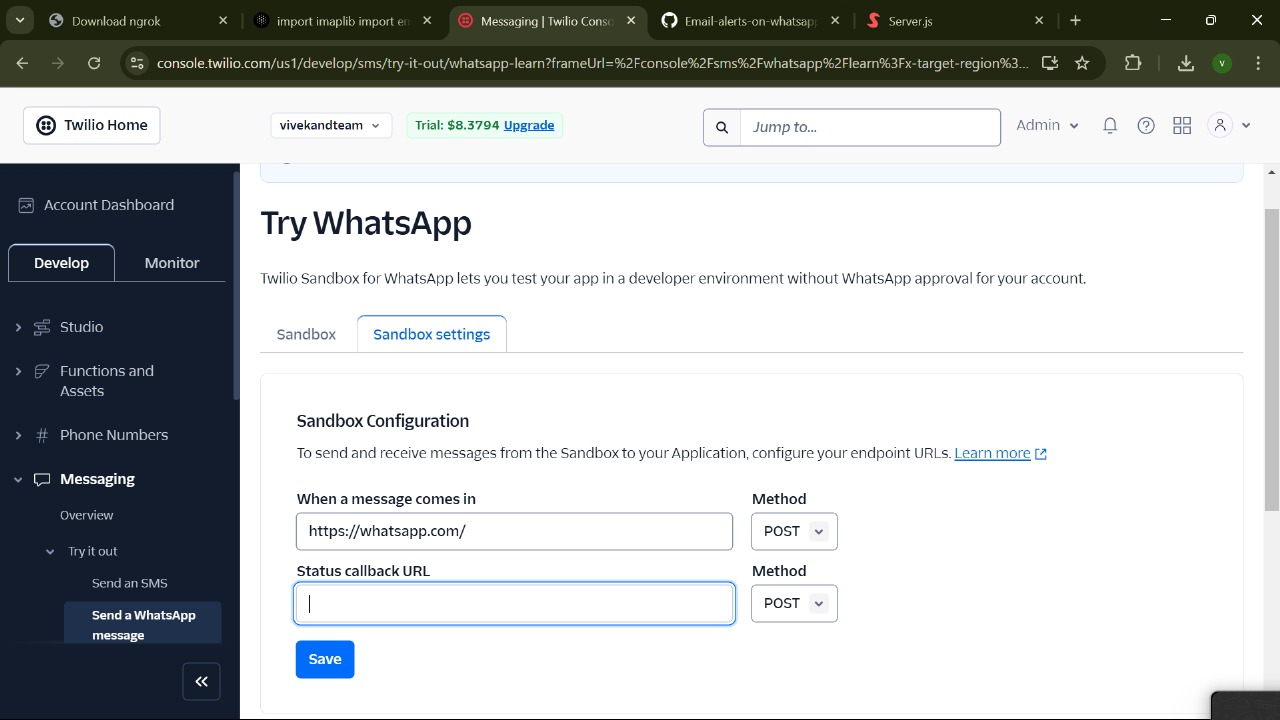
1. Starting a local service
2. Installing the ngrok agent in your terminal
3. Connecting your agent to your ngrok account
4. Start ngrok
5. copy the ngrok server link and paste in the twilio api



**FIG.4.3: ngrok server setup**

1. **Twilio Sandbox URL Setup:**

People around the world use WhatsApp to engage with businesses, and your use case might involve replying to incoming messages from end users. You can use the Twilio Sandbox to explore replying to incoming WhatsApp messages. When an end user sends you a WhatsApp message, Twilio sends a Webhook(a request to a URL that you specify) to your application. In the Sandbox, you can configure that webhook URL in the When a Message Comes infield:



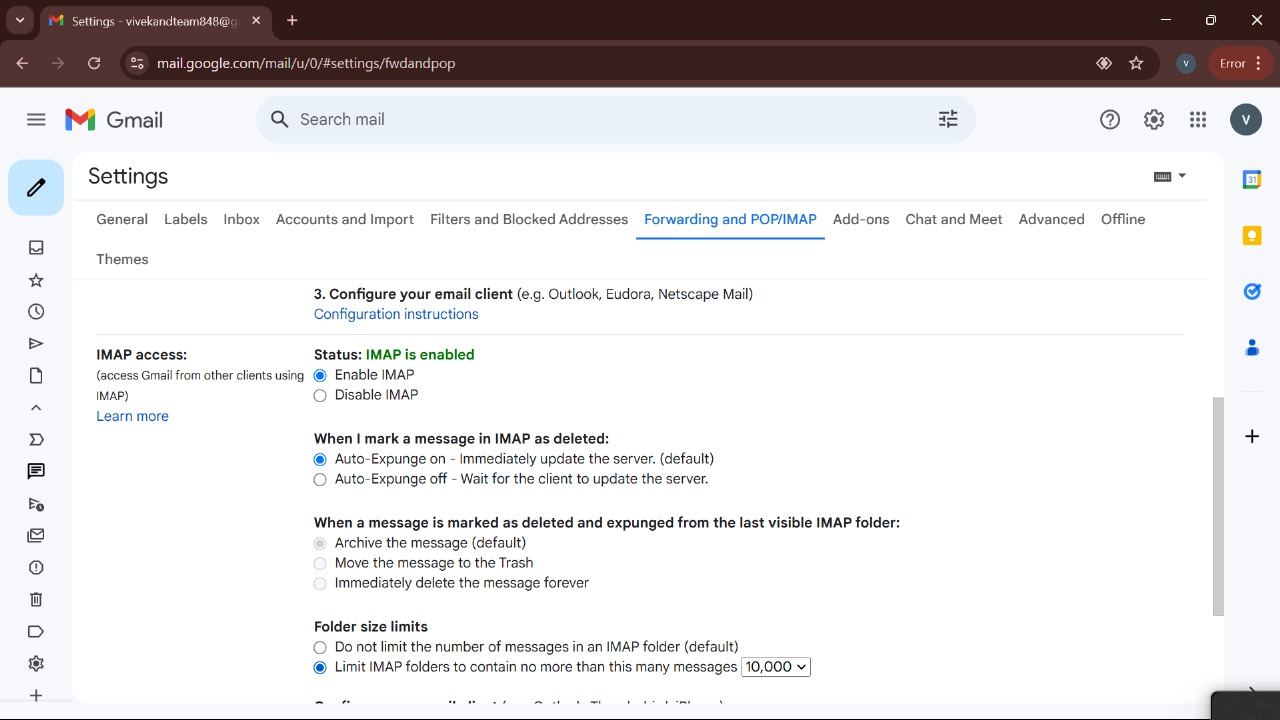
**Fig.4.4: Twilio whatsapp sanbox setup**

1. **Turning on the imap in the gmail:**

Set up IMAP and change your SMTP settings to read Gmail messages in other mail clients, like Microsoft Outlook and Apple Mail. When you use IMAP, you can read yourGmail messages on multiple devices, and messages are synced in real time. You canalso [read Gmail messages using](https://support.google.com/mail/answer/7104828) [POP](https://support.google.com/mail/answer/7104828).

Important: To avoid temporarily locking yourself out of your account, make sure you don't exceed 2500 MB per day for IMAP downloads and 500 MB per day for IMAP uploads.

1. On your computer, open [Gmail](https://mail.google.com/).
2. In the top right, click Settings See all settings.
3. Click the Forwarding and POP/IMAP tab.
4. In the "IMAP access" section, select Enable IMAP.
5. Click Save Changes



**Fig.4.5: Gmail Imap setup**

1. **Setting up the App password for the Gmail**

**Sign in using app passwords:**

An app password is a 16-digit passcode that gives a non-Google app or device permission to access your Google Account. App passwords can only be used with accounts that have [2-Step Verification](https://support.google.com/accounts/answer/185839) turned on.

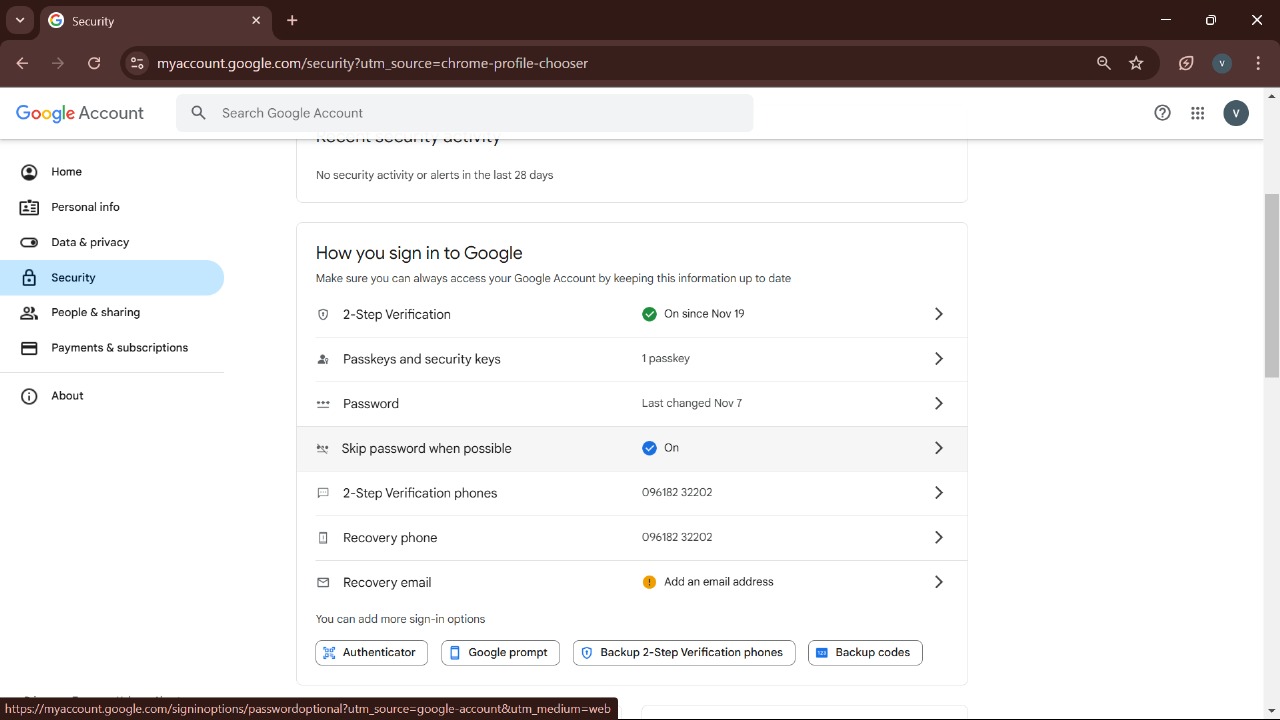
**When to use app passwords:**

To help keep your account secure, use 'Sign in with Google' to connect apps to your Google Account. If the app that you’re using doesn’t offer this option, you can either

* + Use app passwords to connect to your Google Account
  + Switch to a more secure app Create and use app passwords

**Note:** If you use [2-Step-Verification](https://support.google.com/accounts/answer/185839) and are seeing a 'password incorrect' error when trying to access your Google Account, an app password may solve the problem by:

1. Go to your [Google Account](https://myaccount.google.com/).
2. On the left navigation panel, choose Security.
3. On the 'Signing in to Google' panel, choose App passwords. If you don’t see this option:
   * 2-Step Verification is not set up for your account
   * 2-Step Verification is set up for security keys only
   * Your account is through work, school or other organization.
   * You’ve turned on Advanced Protection for your account
4. At the bottom, choose Select app and choose the app that you’re using.
5. Choose Select device and choose the device that you're using.
6. Choose Generate.
7. Follow the instructions to enter the app password. The app password is the 16- character code in the yellow bar on your device
8. Choose Done



**Fig.4.6: Setting up Gmail app password**

**5)Source code**

**i)for fetching unseen mails**

import imaplib

import email

from email.header import decode\_header

from twilio.rest import Client

import time

import schedule

# Twilio configuration

account\_sid = 'AC7aff6c2f221e608eceed18a943ff34c1'  # Replace with your Twilio Account SID

auth\_token = '760e245b34f89d66dcc4d77cdff3e2c5'      # Replace with your Twilio Auth Token

client = Client(account\_sid, auth\_token)

twilio\_whatsapp\_number = 'whatsapp:+14155238886'  # Replace with your Twilio WhatsApp number

recipient\_phone\_number = 'whatsapp:+91 xxxxxxxxxx'  # Replace with the recipient's WhatsApp number

# Email configuration

EMAIL = 'vivekandteam848@gmail.com'  # Replace with your email address

PASSWORD = 'mavg ksqr cmiu kncl'   # Replace with your app password (not your regular password)

IMAP\_SERVER = 'imap.gmail.com'   # Change if using a different provider

def check\_email():

    try:

        # Connect to the email server

        mail = imaplib.IMAP4\_SSL(IMAP\_SERVER)

        mail.login(EMAIL, PASSWORD)

        # Select the mailbox you want to check

        mail.select('inbox')

        # Search for all unseen emails

        status, messages = mail.search(None, 'ALL')  # Change 'ALL' back to 'UNSEEN' to check only unseen emails

        email\_ids = messages[0].split()

        for email\_id in email\_ids:

            # Fetch the email by ID

            res, msg = mail.fetch(email\_id, '(RFC822)')

            msg = email.message\_from\_bytes(msg[0][1])

            # Decode email subject

            subject, encoding = decode\_header(msg['Subject'])[0]

            if isinstance(subject, bytes):

                subject = subject.decode(encoding if encoding else 'utf-8')

            # Get the sender email

            sender = msg['From']

            # Get the email body

            if msg.is\_multipart():

                body = ""

                for part in msg.walk():

                    if part.get\_content\_type() == "text/plain":

                        body = part.get\_payload(decode=True).decode()

            else:

                body = msg.get\_payload(decode=True).decode()

            # Prepare the message to send via WhatsApp

            message\_body = f"From: {sender}\nSubject: {subject}\n\n{body}"

            send\_whatsapp\_message(message\_body)

        mail.logout()

    except Exception as e:

        print(f"An error occurred: {e}")

def send\_whatsapp\_message(body):

    try:

        message = client.messages.create(

            body=body,

            from\_=twilio\_whatsapp\_number,

            to=recipient\_phone\_number

        )

        print(f'Message sent: {message.sid}')

    except Exception as e:

        print(f"Failed to send message: {e}")

# Schedule the email check every minute

schedule.every(1).minutes.do(check\_email)

print("Checking for new emails...")

try:

    while True:

        schedule.run\_pending()

        time.sleep(1)  # Sleep for 1 second

except KeyboardInterrupt:

    print("Email checking stopped by user.")

**ii)for fetching mails and sending to a group/team members**

import imaplib

import email

from email.header import decode\_header

import twilio

from twilio.rest import Client

import time

import schedule

# Twilio configuration

account\_sid = 'AC7aff6c2f221e608eceed18a943ff34c1'  # Replace with your Twilio Account SID

auth\_token = '760e245b34f89d66dcc4d77cdff3e2c5'      # Replace with your Twilio Auth Token

client = Client(account\_sid, auth\_token)

twilio\_whatsapp\_number = 'whatsapp:+14155238886'  # Replace with your Twilio WhatsApp number

# List of recipient phone numbers

recipient\_phone\_numbers = [

    'whatsapp:+91xxxxxxxxxx',  # Replace with the first recipient's WhatsApp number

    'whatsapp:+91xxxxxxxxxx' # Add more numbers as needed

]

# Email configuration

EMAIL = 'vivekandteam848@gmail.com'  # Replace with your email address

PASSWORD = 'mavg ksqr cmiu kncl'   # Replace with your app password (not your regular password)

IMAP\_SERVER = 'imap.gmail.com'   # Change if using a different provider

def check\_email():

    try:

        # Connect to the email server

        mail = imaplib.IMAP4\_SSL(IMAP\_SERVER)

        mail.login(EMAIL, PASSWORD)

        # Select the mailbox you want to check

        mail.select('inbox')

        # Search for all unseen emails

        status, messages = mail.search(None, 'UNSEEN')  # Change 'ALL' back to 'UNSEEN' to check only unseen emails

        email\_ids = messages[0].split()

        for email\_id in email\_ids:

            # Fetch the email by ID

            res, msg = mail.fetch(email\_id, '(RFC822)')

            msg = email.message\_from\_bytes(msg[0][1])

            # Decode email subject

            subject, encoding = decode\_header(msg['Subject'])[0]

            if isinstance(subject, bytes):

                subject = subject.decode(encoding if encoding else 'utf-8')

            # Get the sender email

            sender = msg['From']

            # Get the email body

            if msg.is\_multipart():

                body = ""

                for part in msg.walk():

                    if part.get\_content\_type() == "text/plain":

                        body = part.get\_payload(decode=True).decode()

            else:

                body = msg.get\_payload(decode=True).decode()

            # Prepare the message to send via WhatsApp

            message\_body = f"From: {sender}\nSubject: {subject}\n\n{body}"

            send\_whatsapp\_message(message\_body)

        mail.logout()

    except Exception as e:

        print(f"An error occurred: {e}")

def send\_whatsapp\_message(body):

    try:

        for recipient in recipient\_phone\_numbers:

            message = client.messages.create(

                body=body,

                from\_=twilio\_whatsapp\_number,

                to=recipient

            )

            print(f'Message sent to {recipient}: {message.sid}')

    except Exception as e:

        print(f"Failed to send message: {e}")

# Schedule the email check every minute

schedule.every(1).minutes.do(check\_email)

print("Checking for new emails...")

try:

    while True:

        schedule.run\_pending()

        time.sleep(1)  # Sleep for 1 second

except KeyboardInterrupt:

    print("Email checking stopped by user.")

**iii)for filtering out mails**

import imaplib

import email

from email.header import decode\_header

import twilio

from twilio.rest import Client

import time

import schedule

# Twilio configuration

account\_sid = 'AC7aff6c2f221e608eceed18a943ff34c1'  # Replace with your Twilio Account SID

auth\_token = '760e245b34f89d66dcc4d77cdff3e2c5'      # Replace with your Twilio Auth Token

client = Client(account\_sid, auth\_token)

twilio\_whatsapp\_number = 'whatsapp:+14155238886'  # Replace with your Twilio WhatsApp number

# List of recipient phone numbers

recipient\_phone\_numbers = [

    'whatsapp:+91 xxxxxxxxxx ',  # Replace with the first recipient's WhatsApp number

    'whatsapp:+91 xxxxxxxxxx ',  # Add more numbers as needed

    'whatsapp:+91 xxxxxxxxxx'   # Another example

]

# Email configuration

EMAIL = 'vivekandteam848@gmail.com'  # Replace with your email address

PASSWORD = 'mavg ksqr cmiu kncl'   # Replace with your app password (not your regular password)

IMAP\_SERVER = 'imap.gmail.com'   # Change if using a different provider

# Specify the sender's email address to filter

specific\_sender = 'vivekmandila2223@gmail.com'  # Replace with the sender's email address you want to filter

def check\_email():

    try:

        # Connect to the email server

        mail = imaplib.IMAP4\_SSL(IMAP\_SERVER)

        mail.login(EMAIL, PASSWORD)

        # Select the mailbox you want to check

        mail.select('inbox')

        # Search for unseen emails from a specific sender

        status, messages = mail.search(None, f'(ALL FROM "{specific\_sender}")')

        email\_ids = messages[0].split()

        for email\_id in email\_ids:

            # Fetch the email by ID

            res, msg = mail.fetch(email\_id, '(RFC822)')

            msg = email.message\_from\_bytes(msg[0][1])

            # Decode email subject

            subject, encoding = decode\_header(msg['Subject'])[0]

            if isinstance(subject, bytes):

                subject = subject.decode(encoding if encoding else 'utf-8')

            # Get the sender email

            sender = msg['From']

            # Get the email body

            if msg.is\_multipart():

                body = ""

                for part in msg.walk():

                    if part.get\_content\_type() == "text/plain":

                        body = part.get\_payload(decode=True).decode()

            else:

                body = msg.get\_payload(decode=True).decode()

            # Prepare the message to send via WhatsApp

            message\_body = f"From: {sender}\nSubject: {subject}\n\n{body}"

            send\_whatsapp\_message(message\_body)

        mail.logout()

    except Exception as e:

        print(f"An error occurred: {e}")

def send\_whatsapp\_message(body):

    try:

        for recipient in recipient\_phone\_numbers:

            message = client.messages.create(

                body=body,

                from\_=twilio\_whatsapp\_number,

                to=recipient

            )

            print(f'Message sent to {recipient}: {message.sid}')

    except Exception as e:

        print(f"Failed to send message: {e}")

# Schedule the email check every minute

schedule.every(1).minutes.do(check\_email)

print("Checking for new emails...")

try:

    while True:

        schedule.run\_pending()

        time.sleep(1)  # Sleep for 1 second

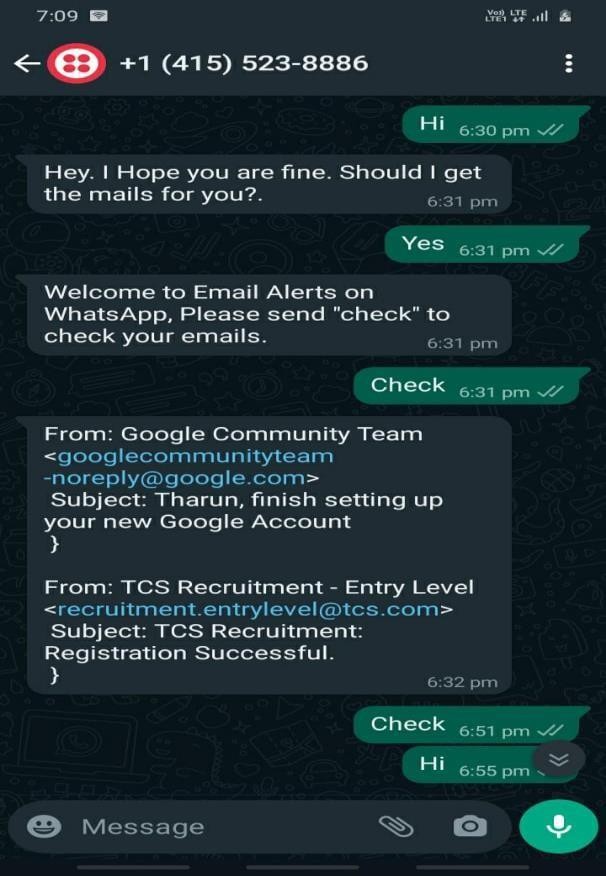
except KeyboardInterrupt:

    print("Email checking stopped by user.")

### CHAPTER 5

### RESULTS AND DISCUSSION

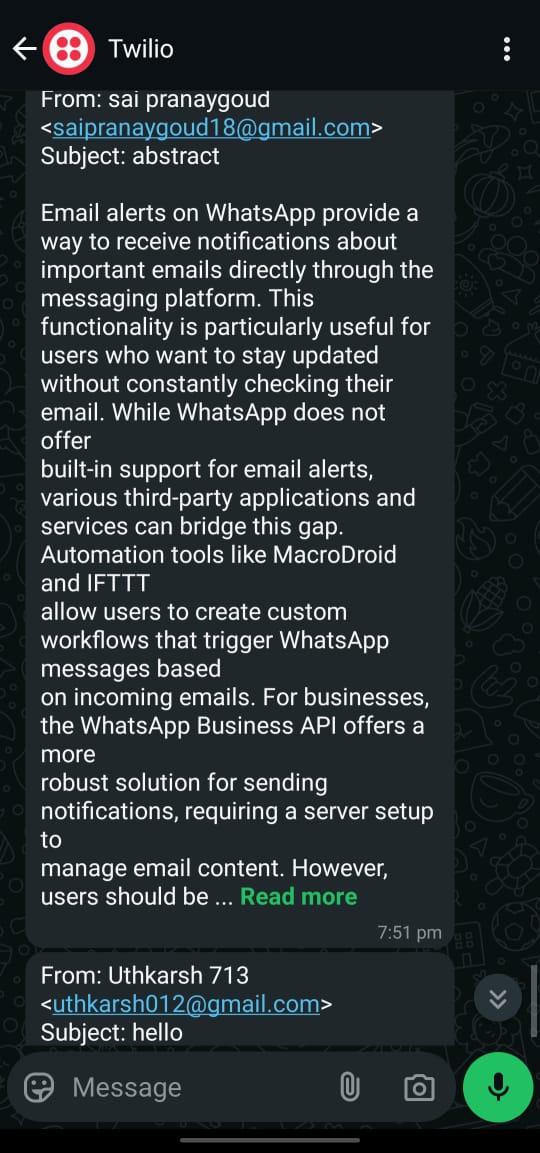
**Result Screenshots**



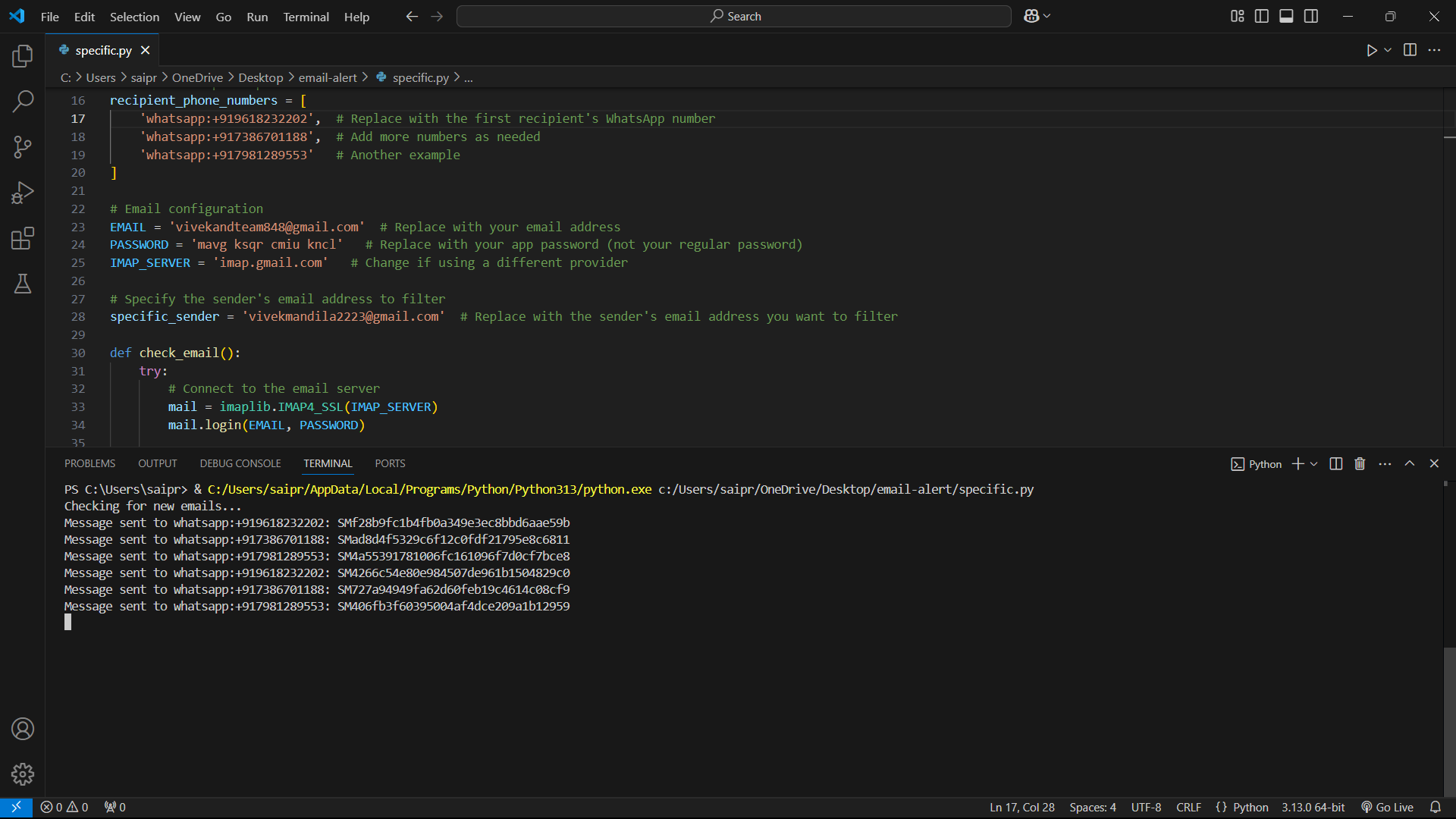
**Fig.5.1: Screen shot of whatsapp bot commands**

Email alerts on WhatsApp using Twilio Sandbox offer real-time notifications, ensuring that you never miss an important message or update. With instant alerts sent directly to your phone, you can stay on top of your work and respond quickly to urgent matters.

We have explored the world of email alerts on WhatsApp using Twilio Sandbox. We learned what Twilio Sandbox is and how it enables developers to build and test their applications with a virtual phone number. We also provided step-by-step instructions on how to set up email alerts on WhatsApp using Twilio Sandbox, as well as the benefits and best practices for using this feature.



**FIg.5.2: WhatsApp bot fetching unseen mails**

****

**Fig.5.3: reading mails from the terminal**

### CHAPTER 6

* 1. **CONCLUSION**

The "Email Alerts on WhatsApp" project has successfully transformed the way critical notifications are delivered and managed, marking a significant advancement in modern communication systems. By seamlessly integrating email alerts with the WhatsApp platform, this project has ushered in a new era of real-time responsiveness and user-centric alerting. The results have been remarkable, with email alerts consistently converted and delivered as WhatsApp messages in real-time, dramatically reducing the time it takes for users to receive vital information.

User adoption rates have been exceptional, as individuals increasingly prefer the immediacy and convenience of WhatsApp notifications over traditional email. Furthermore, the project's emphasis on customization has allowed users to tailor their alert preferences, mitigating notification fatigue and improving engagement. The user experience has seen a significant boost, with high satisfaction ratings underscoring the project's success.

Economically, the project has generated cost savings by optimizing alert delivery processes and reducing reliance on email servers. The ability to interact with alerts in real-time, such as acknowledging critical notifications, has further enhanced the effectiveness of the system. While the project has achieved substantial success, it also opens the door to continuous improvement and future opportunities, including expanding integration with additional sources, exploring advanced notification features, and ensuring ongoing security and privacy compliance. In sum, "Email Alerts on WhatsApp" stands as a testament to the power of innovation in revolutionizing communication and alerting systems, with its impact resonating across various domains.

* 1. **FUTURE SCOPE**

In terms of future work for the "Email Alerts on WhatsApp" project, here are a few potential areas to consider:

**Advanced Message Filtering**: Enhance the system's filtering capabilities to allow users to customize which email alerts they want to receive on WhatsApp based on specific criteria, such as sender, subject, keywords, or importance level. This will provide users with more control over the types of alerts they receive and help reduce information overload.

**Two-Way Communication:** Explore the possibility of enabling two-way communication between users and the email system through WhatsApp. This could involve allowing users to reply to email alerts directly from WhatsApp, with the system intelligently routing the responses back to the appropriate email recipients

**Multi-Language Support**: Extend the system's capabilities to support multiple languages and character sets, allowing users to receive email alerts in their preferred language on WhatsApp. This would require implementing language detection and translation mechanisms to ensure accurate message delivery.

**Attachments and Rich Media Support:** Currently, the focus is on delivering the text content of email alerts on WhatsApp. In the future, consider expanding the system to support attachments and rich media formats, such as images, documents,or voice messages, enhancing the user experience and enabling more comprehensive communication.

### BIBILOGRAPHY

**[1]** Crio Projects – Email Alerts on WhatsApp [https://www.crio.do/projects/javascript-whatsapp-email-alerts/](http://www.crio.do/projects/javascript-whatsapp-email-alerts/)

**[2]** Twilio Docs For WhatsApp Automation

<https://www.twilio.com/docs/whatsapp/tutorial/send-whatsapp-notification-messages-> templates

**[3]** parts of a Highly Converting Email – Analysis for Optimization by Nitin

**[4]** <https://vwo.com/blog/perfect-marketing-email>

**[5]** <https://github.com/nikhilkumarsingh/WhatsAppBotTut>

**[6]** <https://docs.python.org/3/library/imaplib.html>

**[7]** <https://www.tutorialspoint.com/send-mail-from-your-gmail-account-using-python>

**[8]** <https://seaborn.pydata.org/generated/seaborn.countplot.html>

**[9]** <https://www.jetbrains.com/pycharm/download/>

**[10]** GitHub Link [https://github.com/attell](%20https://github.com/attell)