**A**

**Mini Project Report**

**on**

# EMAIL-ALERT VIA WHATSAPP

Submitted to

**Jawaharlal Nehru Technological University, Hyderabad.**

*For the partial fulfillment of requirements for the award of the degree in*

**BACHELOR OF TECHNOLOGY**

in

**COMPUTER SCIENCE AND ENGINEERING**

Submitted by

**MD. AMEER KHAN (22275A0210)**

Under the Esteemed guidance of

**M. SAI PRASAD**

Associate Professor Dept of CSE

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**JYOTHISHMATHI INSTITUTE OF TECHNOLOGY & SCIENCE**

**(Autonomous, NBA (CSE, ECE, EEE) and NAAC 'A' Grade)**

**(Approved by AICTE, New Delhi, Affiliated to JNTUH, Hyderabad)**

**Nustulapur, Karimnagar 505481, Telangana, India**

**2024-2025**

JYOTHISHMATHI
INSTITUTE OF TECHNOLOGY AND SCIENCE
AUTONOMOUS

NAAC 'A' GRADE

NBA NATIONAL BOARD OF ACCREDITATION CSE . ECE . EEE

# <u>CERTIFICATE</u>

This is to certify that the Mini Project Report entitled " **EMAIL-ALERT VIA WHATSAPP** " is being submitted by **MD. AMEER KHAN (22275A0210)** in partial fulfillment of the requirements for the award of the Degree of **Bachelor of Technology** in **Computer Science & Engineering** to the **Jyothishmathi Institute of Technology & Science,** Karimnagar, during academic year 2024-2025, is a bonafide work carried out by him under my guidance and supervision.

The results presented in this Project Work have been verified and are found to be satisfactory. The results embodied in this Project Work have not been submitted to any other University for the award of any other degree or diploma.

| | |
|---|---|
| **Project Guide** | **Head of the Department** |
| **M. SAI PRASAD** | **Dr.R.Jegadeesan** |
| **Associate Professor** | **Professor & HOD** |
| **Dept. of CSE** | **Dept. of CSE** |

**EXTERNAL EXAMINER**

# ACKNOWLEDGEMENT

I would like to express my sincere gratitude to our advisor, **M. Sai Prasad**, whose knowledge and guidance has motivated me to achieve goals I never thought possible. The time I spent working under her/his supervision has truly been a pleasure.

The experience from this kind of work is great and will be useful to me in future. I thank **DR. R. JEGADEESAN**, Professor & HOD CSE Dept for his effort, kind cooperation, guidance and encouraging me to do this work and also for providing the facilities to carry out this work.

It is a great pleasure to convey my thanks to **DR. T. ANIL KUMAR**, Principal, Jyothishmathi Institute of Technology & Science and the College Management for permitting me to undertake this project and providing excellent facilities to carry out my project work.

I thank all the **Faculty** members of the Department of Computer Science & Engineering for sharing their valuable knowledge with me. I extend my thanks to the **Technical Staff** of the department for their valuable suggestions to technical problems.

Finally Special thanks to my parents for their support and encouragement throughout my life and this course Thanks to all my friends and well-wishers for their constant support.

# DECLARATION

I hereby declare that the work which is being presented in this dissertation entitled, "**EMAIL-ALERT VIA WHATSAPP** ", submitted towards the partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering, Jyothishmathi Institute of Technology & Science**, Karimnagar is an authentic record of my own work carried out under the supervision of **M. Sai Prasad, Associate Professor, Department of CSE,** Jyothishmathi Institute of Technology and Science, Karimnagar

To the best of my knowledge and belief, this project bears no resemblance with any report submitted to JNTUH or any other University for the award of any degree or diploma.

**SUBMITTED BY**

**MD. AMEER KHAN (22275A0210)**

**Date:**

**Place:** Karimnagar

# ABSTRACT

This project aims to develop a system that sends automatic email alerts to users via WhatsApp using APIs such as Twilio, enhancing the accessibility and immediacy of email notifications. The system will monitor the user's email account using protocols like IMAP or POP3 to detect incoming emails. A filtering mechanism will be implemented to identify important emails based on criteria such as sender, subject, and keywords. The integration with WhatsApp will be achieved using the Twilio API, which facilitates the sending of real-time notifications directly to the user's WhatsApp account. These notifications will include key details of the email, such as the sender, subject, and a brief snippet of the content. A user-friendly interface will be developed for users to set up and manage email filters, view alert logs, and configure notification preferences. The expected outcome is a robust system that ensures critical emails are promptly delivered to users via WhatsApp, significantly enhancing their ability to respond quickly and manage their communications effectively. This integration aims to bridge the gap between traditional email and modern instant messaging, providing users with a seamless and efficient communication experience. Through this project, users can benefit from the immediacy of WhatsApp notifications, ensuring they never miss important emails.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

| | | |
|---|---|---|
| API | - | Application Programming Interface |
| UI | - | User Interface |
| HTML | - | Hyper Text Markup Language |
| CSS | - | Cascading Style Sheets |
| JS | - | JavaScript |
| OAuth | - | Open Authorization |
| DB | - | Database |
| JSON | - | JavaScript Object Notation |
| REST | - | Representational State Transfer |
| MFA | - | Multi-Factor Authentication |
| RBAC | - | Role-Based Access Control |
| NLP | - | Natural Language Processing |
| NoSQL | - | Not Only SQL |
| E2E | - | End-to-End |
| FIFO | - | First In, First Out |
| CI/CD | - | Continuous Integration / Continuous Deployment |
| UML | - | Unified Modelling Language |

# CHAPTER-1
# INTRODUCTION

## 1.1 PROJECT OVERVIEW

In today’s fast-paced digital world, timely access to crucial information is vital for both individuals and organizations. Email continues to be a dominant communication medium, particularly for official and professional exchanges. However, with the overwhelming influx of emails received daily, users often miss out on important messages that require immediate attention. The project titled “Email Alert on WhatsApp Using Twilio” addresses this challenge by creating a real-time alert system that bridges the gap between email and instant messaging.

By integrating Gmail with WhatsApp, the system ensures that critical emails are not overlooked and that users are promptly notified regardless of their email client or device. The system utilizes the Gmail API to monitor incoming emails and filter them based on user-defined parameters such as sender, subject, or keywords. When an email meeting the specified criteria arrives, a notification is instantly sent to the user’s WhatsApp number using Twilio’s WhatsApp API. The backend is developed using Node.js and Express.js, which handles email parsing, message queuing, and API integration.

This project not only provides a practical demonstration of real-time communication using asynchronous APIs but also showcases the potential of cloud-based automation in enhancing responsiveness. The platform is secure, scalable, and highly customizable, making it suitable for professionals, businesses, and support teams who cannot afford to miss time-sensitive communications. This solution brings numerous benefits, including reduced email-checking fatigue, improved response times, and enhanced productivity. It is especially valuable for individuals like corporate executives, remote workers, and customer service agents who rely on instant communication.

In the future, the system can be expanded to include support for multiple email platforms, a user-friendly dashboard, and advanced filtering using Natural Language Processing (NLP). Overall, “Email Alert on WhatsApp Using Twilio” offers a smart, efficient, and modern approach to staying connected with what truly matters in a user’s inbox.

## 1.2 PROJECT PURPOSE

In today's fast-paced digital landscape, the ability to stay informed about critical email communications is more important than ever. Whether in corporate, academic, or personal settings, timely access to key information can significantly influence decision-making, task execution, and overall responsiveness. However, the reality is that most users receive a high volume of emails daily, making it difficult to constantly monitor their inboxes for time-sensitive or high-priority messages. This challenge becomes even more pronounced for individuals who are frequently on the move or unable to stay connected to their email accounts through traditional desktop or web interfaces.

As a result, important emails may be overlooked or discovered too late, leading to communication breakdowns, missed deadlines, or lost opportunities. On the other hand, instant messaging platforms like WhatsApp have become ubiquitous in modern life, offering real-time, direct, and user-friendly communication on mobile devices. Recognizing the widespread use and reliability of such platforms, this project proposes a novel solution: an automated system that forwards selected, filtered email notifications directly to a user’s WhatsApp account. By integrating the Gmail API for email access and filtering, along with the Twilio WhatsApp API for messaging delivery, the system bridges the gap between email and instant messaging.

Users can define filters based on sender, subject line, keywords, or other parameters, ensuring that only important emails are forwarded. The backend system, built using Node.js and Express.js, continuously monitors the inbox, applies the defined filters, and formats the relevant email content into a concise, readable WhatsApp message. This message may include the sender's name, subject, timestamp, and a short snippet of the email body, allowing the user to quickly assess the email's importance and act accordingly.

By delivering these alerts in real-time, the system minimizes the risk of overlooking urgent communications, significantly improves user convenience, and enhances productivity. Furthermore, it offers a platform-independent solution accessible from any device with WhatsApp, making it ideal for professionals, remote workers, executives, and anyone who relies on critical email updates.

## 1.3 PROJECT SCOPE

The scope of the “Email Alert on WhatsApp Using Twilio” project encompasses the design, development, and deployment of a real-time notification system that integrates Gmail with WhatsApp. This project aims to improve user awareness of critical email communications by forwarding selected, filtered email alerts directly to WhatsApp. The system focuses on delivering a seamless and automated workflow that allows users to receive important email summaries on their mobile devices, eliminating the need to constantly monitor inboxes. This solution is especially applicable in environments where instant responsiveness is vital, such as in professional, academic, or customer service contexts. The scope also includes handling user-defined filters for email selection, formatting messages for WhatsApp delivery, ensuring secure authentication via Google APIs, and ensuring compatibility across different user devices. Although the initial deployment focuses on Gmail and WhatsApp, the system is designed to be scalable and extendable to support additional email providers and messaging platforms in the future.

## 1.4 PROJECT FEATURES

- **Real-Time Email Alerts:** The system provides immediate WhatsApp notifications for selected incoming emails, ensuring critical messages are not missed.
- **Custom Filtering Options:** Users can define filtering rules based on sender address, subject line, or email content. This helps reduce unnecessary alerts and focus on important messages.
- **Secure API Integration:** Authentication is handled securely using OAuth 2.0 with the Gmail API, ensuring user privacy and data protection during email access.
- **Mobile-Friendly Notifications:** WhatsApp-based alerts make the system highly mobile-friendly, allowing users to receive important updates on the go. Efficient.
- **Backend Architecture:** The backend, developed with Node.js and Express.js, ensures fast processing, scalability, and easy integration of additional features.
- **Queue Management and Reliability:** To avoid duplicate or missed alerts, the system manages a queue of processed emails and includes error handling for stable performance.

- **User Independence from Email Clients:** The user doesn’t need to be logged into their Gmail account continuously, as alerts are pushed directly to WhatsApp without manual checks.
- **Extendability for Future Use Cases:** The modular design allows integration with other platforms like Outlook or Telegram, making the system future-ready and flexible.
- **Multi-User Support:** The system can be configured to support multiple users with individual filtering criteria and WhatsApp numbers. This makes it suitable for teams or small organizations needing customized notifications.
- **Compact and Informative Message Format:** Each WhatsApp alert is designed to be concise yet informative, including the sender's name, subject line, and a short preview of the email body. This helps users quickly understand the email’s context without opening their inbox.
- **Minimal User Configuration:** Once set up, the system requires little to no manual intervention. Users can update their filter rules or WhatsApp number through a simple configuration file or interface, promoting ease of use and automation.

# CHAPTER-2

# LITERATURE REVIEW

**1. Overview of Email Notification Systems.**

Email notification systems are essential for delivering timely updates to users, especially for critical communications. While platforms like Gmail and Outlook provide useful features such as categorized inboxes and push notifications, these systems are largely confined within their own applications. This limitation reduces their effectiveness in multi-platform communication environments, as users often rely on a variety of messaging platforms in their daily routines.

**2. Limitations in Cross-Platform Communication.**

Despite advancements in email technologies, a significant gap exists in terms of cross-platform integration. Email notifications typically remain restricted to the email app itself, failing to extend their reach to other commonly used messaging services. This creates challenges in ensuring prompt user responses, particularly when high-priority emails are not immediately noticed due to platform boundaries.

**3. Emergence of WhatsApp in Real-Time Notifications.**

WhatsApp has become one of the most popular platforms for real-time communication due to its rapid message delivery, high open rates, and ease of use. Its widespread adoption makes it an ideal candidate for disseminating time-sensitive alerts. With support for automation via APIs like Twilio's, WhatsApp enables a broad range of applications, including account authentication, system notifications, and automated messaging, expanding its utility in notification systems.

**4. Integration of Email Alerts with Messaging Platforms.**

Integrating email services like Gmail with messaging platforms such as WhatsApp offers a powerful solution to the limitations of traditional notification systems. This integration facilitates seamless, cross-platform alert delivery, improving productivity and reducing the likelihood of missed communications. In professional and time-sensitive contexts, such integration ensures that important messages are promptly received and acted upon.

**5. Message Queue Management and Rate.**

Limiting As notification volumes increase, efficient message queue management becomes essential to avoid delays and maintain system performance. Rate limiting especially when interacting with APIs like Twilio is a critical consideration. Implementing structured queue processing and throttling mechanisms ensures that notifications are sent reliably without exceeding service limits or compromising system responsiveness.

**6. Security and Privacy Considerations.**

Forwarding email content to external messaging platforms introduces important concerns related to data security and user privacy. Sensitive information must be protected using encryption, data anonymization, and secure transmission protocols. Ensuring compliance with data protection standards is vital to maintaining user trust and safeguarding confidential content during integration between systems.

**7. Enhancing User Engagement Through Familiar Platforms.**

User responsiveness is greatly improved when notifications are delivered through platforms they frequently use. Messaging apps like WhatsApp offer a more immediate and engaging experience than traditional email alerts. Utilizing these platforms for real-time notifications leads to quicker user actions, increased visibility for critical messages, and an overall enhancement in user engagement especially in environments where timing is crucial.

Email notification systems are a crucial component of modern digital communication, ensuring users remain informed of important updates. However, current platforms like Gmail and Outlook primarily offer native push alerts, web notifications, and categorized email tabs such as Primary, Promotions, and Social. These systems remain confined to the email ecosystem and lack integration with widely-used messaging platforms like WhatsApp, leading to a gap in real-time accessibility and responsiveness. In their study "Limitations of Traditional Email Notification Systems and the Need for Cross-Platform Alert Integration", Kumar and Singh (2019) stress the importance of cross-platform integration to ensure timely user attention to critical emails. Supporting this, John and Smith (2018) emphasized that traditional systems fail to support actionable, real-time notifications necessary for high-priority communications [1][7].

In contrast, WhatsApp has emerged as a superior platform for real-time notification delivery. With its high open and response rates, WhatsApp is increasingly adopted in alert dissemination systems. According to the study by Williams et al. (2020) titled “Effectiveness of WhatsApp in Alert Dissemination”, WhatsApp significantly outperforms email in user engagement due to its immediacy and familiarity [2].

The availability of Twilio’s WhatsApp API enables seamless integration of programmable messaging for use cases like authentication, customer alerts, and automated messaging, strengthening its viability for alert systems. Efforts to bridge email systems with WhatsApp are being actively explored. In “Cross-Platform Communication: Integrating Email and Messaging Applications”, Clarke and Doe (2021) demonstrated that combining Gmail notifications with platforms like WhatsApp greatly improves user responsiveness, especially in enterprise or time-sensitive scenarios. This kind of integration helps mitigate delays, reduce missed communications, and streamline cross-platform interactions [3].

However, scaling such systems introduces technical complexities, particularly when managing message queues and adhering to API rate limits. Li, Chen, and Kumar (2019) in “Queue Management and Rate Limiting in High Volume Notification Systems” proposed robust strategies to avoid message delivery delays caused by Twilio’s constraints. Their work is particularly valuable for real-time applications requiring guaranteed delivery and order of notifications [4]. On the other hand, integrating third-party platforms like WhatsApp with Gmail raises privacy and data protection concerns. In their paper “Privacy Concerns in Multi-Platform Notification Systems”, Brown and Lee (2020) cautioned about forwarding confidential information and recommended employing end-to-end encryption, anonymization, and limited data exposure practices [5].

These privacy-preserving approaches are crucial for building trust and ensuring compliance in enterprise and user-level applications. The impact of using platforms like WhatsApp on user engagement and response time is also well-documented. In “Enhancing User Engagement with Instant Messaging Alerts”, Johnson (2019) found that WhatsApp-based alerts led to substantially faster responses compared to traditional email notifications. Familiarity and frequency of use were cited as major factors influencing this improvement [6].

# CHAPTER -3
# EXISTING & PROPOSED SYSTEM

## 3.1 EXISTING SYSTEM

Current email systems, such as Gmail and Outlook, offer native notifications through mobile and web alerts, along with categorized inboxes like Primary, Promotions, and Social. However, these notifications are limited to the email platform and do not extend to widely-used messaging apps like WhatsApp, which many users prefer for real-time communication. Though some platforms provide SMS alerts as an alternative, these come with notable drawbacks—limited content, lack of interactivity, potential delivery delays, and additional costs. SMS cannot match the responsiveness or rich features of modern messaging platforms. These limitations highlight the need for cross-platform integration of email notifications with messaging services like WhatsApp. Such integration would enhance real-time alert delivery, improve user engagement, and minimize the chances of missing critical emails. Leveraging platforms like WhatsApp, known for high open rates and interactive capabilities, enables the development of smarter, more efficient, and user-friendly notification systems tailored to today's communication preferences.

## 3.2 EXISTING SYSTEM DISADVANTAGES

1. **Platform Restriction:**
   Users are confined to receiving notifications only within the email platform (e.g., Gmail app) or device notifications, which may be easy to miss due to notification overload.
2. **Lack of Real-Time Customization:**
   Existing systems typically do not offer the ability to filter specific types of alerts or selectively forward alerts from particular email senders. Users are unable to customize notifications based on priority emails or keywords within the subject line.
3. **Inadequate Cross-Platform Integration:**
   Although some integrations exist (e.g., using third-party automation tools like IFTTT or Zapier), these solutions are often complicated, have limited customization, and may require multiple subscriptions or accounts. This limits seamless, real-time integration with WhatsApp for non- technical users.

4. **Security and Privacy Concerns:**
   Native email notification systems may not prioritize advanced security or privacy settings when forwarding data to other platforms. Sensitive data shared through insecure channels can pose privacy risks.
5. **No Message Queue Handling:**
   Existing systems do not incorporate a message queue for handling multiple notifications or managing delivery delays due to rate limits (e.g., Twilio's restrictions on WhatsApp messages). This can lead to missed notifications when high-priority emails are received in succession.

## 3.3 PROBLEM STATEMENT

In the current college event management landscape, the prevalent manual processes pose significant challenges, resulting in inefficiencies and communication gaps. Coordinators face difficulties in tracking event details and managing participant enrollments, while participants often find it inconvenient to access real-time information about events, leading to delayed registrations. The lack of a centralized platform exacerbates these issues, hindering seamless communication between organizers and participants. Administrative tasks, such as event creation and user account management, become cumbersome without a streamlined digital system. To address these challenges, the proposed College Event Management System aims to automate and simplify the entire event lifecycle. By leveraging modern technologies, the system endeavors to enhance efficiency, reduce manual efforts, and provide a user-friendly experience for both organizers and participants. The implementation of real-time communication channels and a robust reporting system is anticipated to contribute significantly to the effectiveness of event planning and execution within the college community.

## 3.4 PROPOSED SYSTEM

The proposed system aims to deliver a reliable, real-time alert system for email notifications via WhatsApp, ensuring users receive only essential messages. This system leverages the Gmail API for email access, Twilio's WhatsApp API for message delivery, and implements a priority-based message queue for optimized notification handling. By integrating these technologies, the system provides secure, customized, and seamless notifications that users can control and tailor to their needs.

**Key Features of the Proposed System:**

1. **Selective Email Alerts via WhatsApp.**

   Users can define specific email senders or keywords in email subjects to trigger notifications on WhatsApp, allowing them to focus on high-priority messages without being overloaded by every incoming email. This feature is customizable, ensuring that only essential information reaches the user in real time.

2. **Real-Time Notification Delivery.**

   The proposed system polls Gmail at regular intervals and checks for new, relevant emails. When an email from a specified sender or with specific criteria is detected, it immediately generates a WhatsApp message, ensuring prompt delivery.

3. **Message Queue with Rate-Limit Management.**

   By implementing a message queue system, the proposed solution efficiently handles Twilio's WhatsApp rate limits, ensuring that no messages are missed. If Twilio's rate limit is reached, the system will automatically delay and retry the message delivery to maintain reliable communication.

4. **Secure Authentication and Access.**

   Using OAuth2.0, the proposed system provides secure access to the Gmail API, ensuring that user data remains protected and email access is granted only by explicit user authorization. The system keeps authentication tokens locally, allowing secure and persistent access to the Gmail account.

5. **User-Friendly Web Interface.**

   A web interface, developed with Express.js and React, allows users to easily manage alert settings, such as adding or removing specific email addresses, adjusting keywords, and testing the alert system. This user-centered design ensures that the system is accessible to both technical and non-technical users.

6. **Rate Limiting and Error Handling.**

   By incorporating error handling and retry logic, the proposed system can handle failed message deliveries due to network issues or rate limits, ensuring that notifications are not missed and that alerts remain accurate and timely.

## 3.5 PROPOSED SYSTEM ADVANTAGES

- **Prioritized Communication Efficiency**

  By enabling users to filter alerts based on sender or keywords, the system ensures that only high-priority emails trigger WhatsApp notifications. This reduces notification fatigue and helps users focus on essential communications without distractions from low-importance messages.

- **Real-Time Responsiveness**

  The polling mechanism for Gmail, combined with instant WhatsApp alert generation, ensures that users are notified as soon as a critical email arrives. This real-time response capability is especially valuable in time-sensitive scenarios like business communication or academic correspondence.

- **Robust Rate-Limit Compliance and Message Reliability**

  With a built-in message queuing system and retry logic, the system intelligently manages Twilio’s API rate limits. It prevents message loss and ensures delivery continuity by automatically handling delayed or failed alerts an essential advantage in high-volume email environments.

- **High-Level Security and Privacy**

  The use of OAuth 2.0 authentication secures user credentials and ensures controlled access to Gmail data. Only explicitly authorized applications can access the inbox, which safeguards user privacy and mitigates the risk of unauthorized data exposure.

- **Enhanced Usability via Web Interface**

  A user-friendly web dashboard simplifies the customization of alert settings. Both technical and non-technical users can easily manage notification criteria, making the system broadly accessible without requiring advanced configuration skills.

- **Platform Independence and Multi-Device**

  Accessibility Since WhatsApp is accessible on both mobile and desktop, users can receive and act on email alerts from anywhere. This cross-platform capability ensures continuous connectivity and faster response times, regardless of the device being used.

- **Scalability and Flexibility**

  The modular design using Gmail API, Twilio API, and React-based UI allows for future enhancements such as multi-user support, analytics, or integration with other services like Slack or Telegram, ensuring long-term system adaptability.

# CHAPTER-4
# SYSTEM REQUIREMENTS

## 4.1 SOFTWARE REQUIREMENTS

Software Requirements specifies the logical characteristics of each interface and software components of the system. The following are the software requirements,

- **Node.js:** Server-side runtime for handling backend logic.
- **MongoDB:** For managing user settings and configuration data.
- **Twilio:** For integration with WhatsApp to send notifications.
- **Gmail API:** For email monitoring.
- **OAuth 2.0:** For secure authentication with Gmail.

## 4.2 HARDWARE REQUIREMENTS

Hardware interfaces specify the logical characteristics of each interface between the software product and the hardware components of the system. The following are some hardware requirements,

1. **Server**:

- Processor: 2.0 GHz or higher
- RAM: 4 GB minimum
- Storage: 10 GB minimum (for application files and database)

2. **Client Device**:

- Any device capable of receiving WhatsApp messages (smartphone with an active WhatsApp account).

## ➢ FUNCTIONAL REQUIREMENTS

### 1. User Registration and Configuration

- Users can register their email address and WhatsApp number for receiving alerts.
- Users can select specific email addresses or filter criteria for receiving alerts.

2. **Email Monitoring**

- The system should connect to the user’s Gmail account using the Gmail API.
- It should periodically check for new emails.
- Only relevant emails, based on user-defined filters (e.g., specific sender), should trigger a WhatsApp alert.

3. **WhatsApp Alert System**

- Integration with Twilio’s WhatsApp API for sending real-time alerts to the user’s WhatsApp.
- Each alert should include the email's subject, sender information, and a link to the email in the user’s Gmail account.
- Retry functionality should be implemented to handle message failures due to network or API rate limitations.

4. **Error Handling and Notification**

- System should handle API errors gracefully, logging any issues encountered during email monitoring or WhatsApp notification attempts.
- Notifications should be retried in case of temporary failures, with exponential backoff if API rate limits are reached.

➢ **NON-FUNCTIONAL REQUIREMENTS**

5. **Reliability**

- The system should handle interruptions in the Gmail and Twilio APIs gracefully.
- Use appropriate error handling to ensure the system doesn’t crash due to API issues.

6. **Scalability**

- The system should handle multiple users without significant performance degradation.
- Queueing systems should be in place to handle the rate limits imposed by the WhatsApp API for message sending.

7. **Security**

- Secure storage of user credentials using OAuth 2.0 for accessing Gmail.
- Authentication tokens should be stored securely and refreshed periodically to maintain secure access to Gmail.

8. **Performance**

- The system should check for new emails at user-configured intervals (e.g., every 10 seconds).
- Minimize API calls to avoid exceeding rate limits, and use caching where applicable to optimize performance.

9. **Usability**

- The system should be easy to configure and operate, with a simple interface for setting up alerts.
- User notifications should be clear, informative, and formatted for easy reading on WhatsApp.

## ➢ TECHNICAL REQUIREMENTS

- **Programming Language**: JavaScript (Node.js for server-side logic)
- **Frontend**: ReactJS for setting up user configurations
- **Backend Framework**: Express.js for handling API requests
- **Database**: MongoDB (for storing user settings and configuration)
- **APIs:**
    1. **Gmail API**: For accessing the user's email data.
    2. **Twilio API**: For sending WhatsApp notifications.

# CHAPTER-5
# PROJECT DESCRIPTION

The Email Alert on WhatsApp Using Twilio system is designed as a multi-layered architecture that seamlessly integrates several technological components to enable real-time email monitoring and alert delivery through WhatsApp. This architecture ensures reliable communication, scalability, security, and ease of use, combining both client-side and server-side functionalities.

## 5.1 MODULES

### 5.1.1 BACKEND/ADMIN DASHBOARD

➢ **User Configuration and Alert Rules Management**

The backend includes a robust Admin Dashboard that allows complete control over user-specific alert configurations. Each user—whether an individual or part of an organization—can have personalized rules that determine which emails should trigger a WhatsApp alert. Admins can create and manage user profiles with the following fields:

- Full Name
- Registered Email Address (Gmail)
- Registered WhatsApp Number
- Authentication Tokens (securely encrypted)
- Alert Preferences (e.g., Keywords, Senders)
- Active/Inactive Rule Status
- Polling Interval (custom time intervals for email checks)

Admins can view a complete list of users along with their activity status, alert logs, and current configurations. The system ensures user configurations are modular and isolated, so each user receives alerts based on their defined parameters without affecting others.

➢ **Email Monitoring & Priority-Based Processing**

A key backend feature is real-time email monitoring through the Gmail API. The admin dashboard allows setting polling intervals, typically between 1–5 minutes, to ensure timely retrieval of new messages. Administrators can configure:

- **Keyword Filtering:** Only trigger alerts if specific words (e.g., “urgent”, “invoice”, “deadline”) are found in subject or body.
- **Sender Whitelist:** Allow alerts only from selected email addresses or domains.
- **Attachment Handling Rules:** Enable alerts only if an email includes attachments (optional).
- **Thread Management:** Avoid duplicate alerts for the same conversation thread.

➢ **Twilio WhatsApp API Integration Settings**

The Admin Panel allows seamless integration and configuration of the Twilio WhatsApp messaging API, which is responsible for delivering alerts. Key configuration options include:

- Twilio Account SID & Auth Token input fields with encryption
- Sender Number (Twilio WhatsApp-approved number)
- Rate Limit Configuration: Define how many messages per minute/hour can be sent
- **Custom Message Templates:** Editable templates using dynamic fields such as:
    - Sender
    - Subject
    - Time Received
    - Message Preview
- **Fallback Messaging Options:** If a message fails, the system can:
    - Retry after a predefined interval
    - Send a fallback alert (e.g., via email or SMS)

Admins can also enable logging of API errors and warnings, helping them proactively resolve issues related to authentication or message failures.

➢ **Notification Queue & Rate Limit Handling**

Since Twilio enforces messaging limits (e.g., 1 message/sec per number), a Message Queue System is implemented to handle:

- **Queue Management:** Emails are placed in a first-in, first-out (FIFO) queue before being sent
- **Retry Mechanisms:** If Twilio’s rate limit is reached, the system waits and retries delivery automatically
- **Status Monitoring:** Each message has a status Pending, Sent, Failed, Retried

- **Alert Prioritization:** Urgent emails can bypass the queue based on flags

This mechanism ensures scalability and reliability even under high volumes of alerts.

- **Keyword Filtering:** Only trigger alerts if specific words (e.g., “urgent”, “invoice”, “deadline”) are found in subject or body.
- **Sender Whitelist:** Allow alerts only from selected email addresses or domains.
- **Attachment Handling Rules:** Enable alerts only if an email includes attachments (optional).
- **Thread Management:** Avoid duplicate alerts for the same conversation thread.

➢ **Twilio WhatsApp API Integration Settings**

The Admin Panel allows seamless integration and configuration of the Twilio WhatsApp messaging API, which is responsible for delivering alerts. Key configuration options include:

- Twilio Account SID & Auth Token input fields with encryption
- Sender Number (Twilio WhatsApp-approved number)
- Rate Limit Configuration: Define how many messages per minute/hour can be sent
- **Custom Message Templates:** Editable templates using dynamic fields such as:
    - Sender
    - Subject
    - Time Received
    - Message Preview
- **Fallback Messaging Options:** If a message fails, the system can:
    - Retry after a predefined interval
    - Send a fallback alert (e.g., via email or SMS)

Admins can also enable logging of API errors and warnings, helping them proactively resolve issues related to authentication or message failures.

➢ **Notification Queue & Rate Limit Handling**

Since Twilio enforces messaging limits (e.g., 1 message/sec per number), a Message Queue System is implemented to handle:

- **Queue Management:** Emails are placed in a first-in, first-out (FIFO) queue before being sent
- **Retry Mechanisms:** If Twilio’s rate limit is reached, the system waits and retries delivery automatically

- **Status Monitoring:** Each message has a status Pending, Sent, Failed, Retried
- **Alert Prioritization:** Urgent emails can bypass the queue based on flags

**System Logs, Delivery Reports, and Analytics**

The dashboard provides in-depth monitoring of the alert system, including:

- Email Logs: Timestamped logs of every email checked, matched, or ignored
- Alert Logs: History of all alerts sent via WhatsApp, with status and delivery time
- Error Logs: Detailed records of errors such as API failures, authentication errors, or message queue overflows
- Analytics Dashboard: Graphs and tables showing:
    - Number of emails received
    - Number of alerts sent
    - User engagement statistics
    - Failed vs Successful messages

Reports can be downloaded in CSV or PDF format for auditing purposes.

### 5.1.2 FRONTEND

➢ **Home Page**

The user-facing frontend is built using React.js with a responsive, intuitive design. The homepage presents:

- A clean summary of:
    - Number of alerts received today/this week.
    - Active alert rules.
    - WhatsApp delivery statuses.
- A visual card/grid layout to show recent alerts.
- Live email activity tracker that shows incoming Gmail messages filtered by the user's rules.
- This dashboard ensures users are always aware of the system's activity and can manage their configurations on the go.

➢ **Alert Rule Configuration Interface**

Users can easily manage their alert preferences from a dedicated configuration page & they can:

- Add new rules with specific sender emails, subject keywords, or attachment flags.

- Toggle individual rules ON or OFF.
- Set DND schedules (e.g., block alerts during 11 PM – 7 AM).
- Set alert frequency (e.g., maximum 1 alert every 15 minutes).
- Customize message format with a preview option.

➢ **Notification Testing and Preview Module**

The system provides a "Test Alert" feature that allows users to:

- Enter a dummy subject/body and test how the alert would appear on WhatsApp
- Validate WhatsApp number integration with Twilio
- Check template rendering and formatting

This helps users ensure their setup is working before real alerts start flowing.

➢ **Notification History and Status Panel**

Users can navigate to the History tab to see:

- Alerts triggered (with date & time).
- Subject lines and sender information.
- Status of WhatsApp delivery (Sent, Pending, Failed).
- Retry logs if applicable.
- Each record can be clicked for detailed metadata and debug info.

➢ **Error Handling and Troubleshooting Assistance.**

- To support users, the system includes a "Help & Logs" section where they can:
- View common errors and fixes (e.g., “Invalid Auth Token”, “Phone Number Not Registered”)
- Re-initiate failed message deliveries.
- View live server health status.

# CHAPTER-6
# SYSTEM DESIGN

## 6.1 SYSTEM ARCHITECTURE

The Email Alert via WhatsApp project consists of multiple components working together to monitor incoming Gmail messages and send alerts through WhatsApp. This system architecture includes the front end (user interface for configuration), backend logic, external APIs, and database integration.

## 6.2 USE CASE DIAGRAM

A Use Case Diagram is a visual representation of the interactions between the users (actors) and a system. It depicts the various use cases or scenarios where the system is used to achieve a goal. In a use case diagram:

**Actors:** Represent external entities (people or systems) that interact with the system. They are usually depicted as stick figures.

**Use Cases**: Represent the functions or actions the system performs in response to the actors' interactions, depicted as ovals.

**Relationships**: Indicate how actors and use cases are connected, often showing the flow of interaction between them.

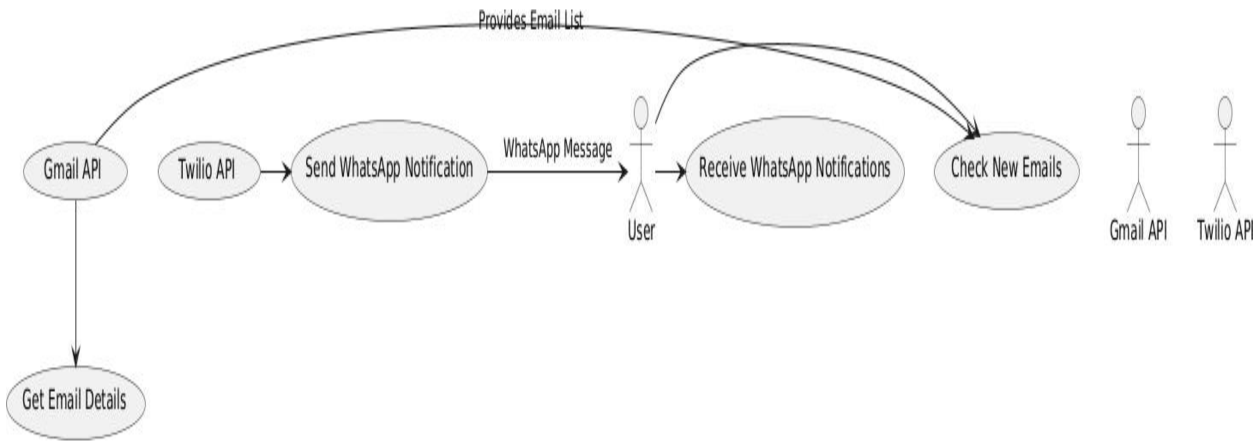


Figure 6.1 Use Case Diagram of the Email-to-WhatsApp Alert Workflow.

**Explanation of the Above Use Case Diagram:**

**Actors:**

- **User:** The main person receiving WhatsApp notifications about new emails.
- **Gmail API**: External system interacting with the application to retrieve email information.
- **Twilio API**: External system that sends WhatsApp notifications.

**Use Cases:**

- **Get Email Details:** This use case is handled by the Gmail API to fetch new email information (subject, sender).
- **Check New Emails:** The system regularly checks for new emails via the Gmail API.
- Send WhatsApp Notification: The Twilio API sends the WhatsApp message to the user, containing the email details.
- **Receive WhatsApp Notifications**: The user receives notifications about new emails on WhatsApp.

**Process Flow:**

- The system checks for new emails by interacting with the Gmail API.
- It retrieves the email details (subject, sender) when a new email is found.
- Using the Twilio API, it sends a WhatsApp notification to the user with the email details.
- The user receives the WhatsApp message and is notified of the new email.

## 6.3 CLASS DIAGRAM

A Class Diagram is a type of static structure diagram in UML (Unified Modeling Language) that describes the system's structure by showing:

- Classes: Represented by rectangles that define the attributes (variables) and operations (methods) of objects.
- Relationships: Lines or arrows between classes show the associations, dependencies, or interactions among the different classes.

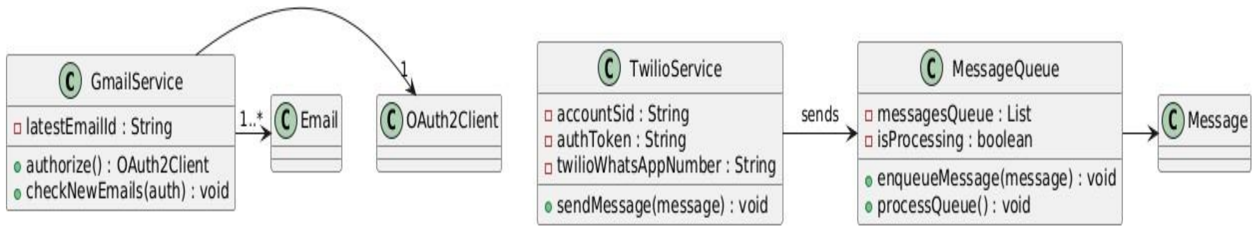


Figure 6.2 Class Diagram of the Email-to-WhatsApp Alert Workflow.

**Explanation of the Above Class Diagram:**

**1. Gmail Service**

Handles Gmail API integration and monitors the user's inbox for new emails.

**Attributes**

- LatestEmailId (*String*): Stores the ID of the most recently fetched email to avoid duplicate alerts.

**Methods**

- authorize() → OAuth2Client:
  Authenticates the application using OAuth 2.0 and returns a valid authorization client.
- Check New Emails(auth:OAuth2Client) → void:

  Uses the authenticated client to access Gmail and check for new, unread emails matching specified filters.

**2. OAuth2Client**

Represents the authentication client for secure Gmail API access via OAuth 2.0.

**Description**

- Encapsulates authorization mechanisms (token handling, refresh logic, etc.) for secure API usage.

**3. Email**

Data model representing an individual email message.

**Attributes**

- sender (*String*)

- subject (*String*)
- timestamp (*Date Time*)
- message Id (*String*)
- body (*String*)

**4. Twilio Service**

Manages Twilio API communication for WhatsApp messaging.

**Attributes**

- accountSid (*String*): Twilio Account SID for authentication.
- authToken (*String*): Twilio API authentication token.
- twilioWhatsAppNumber (*String*): Twilio-provided WhatsApp number for sending messages.

**Methods**

- sendMessage(message: String) → void:

  Sends a formatted WhatsApp message to the user via Twilio API.

**5. Message Queue**

Handles asynchronous message processing with sequential, reliable delivery.

**Attributes**

- messagesQueue (*List<String>*): Queue of pending messages.
- isProcessing (*boolean*): Flag to prevent concurrent processing.

**Methods**

- enqueueMessage(message: String) → void:

  Adds a message to the queue.

- processQueue() → void:
  Processes the queue in FIFO order (ensures rate-limiting and reliability).

## 6.4 ACTIVITY DIAGRAM

An Activity Diagram is a type of behavioral diagram in UML that visually represents the flow of activities or actions within a system or process. It shows:

- **Actions/Activities**: Represented by rectangles or rounded rectangles, indicating tasks performed.
- **Decisions:** Diamond shapes where flow diverges based on conditions.
- **Control Flow**: Arrows that represent the flow from one activity to another.
- **Start and End Nodes**: The beginning is marked by a black circle, and the end by a filled circle with a border.

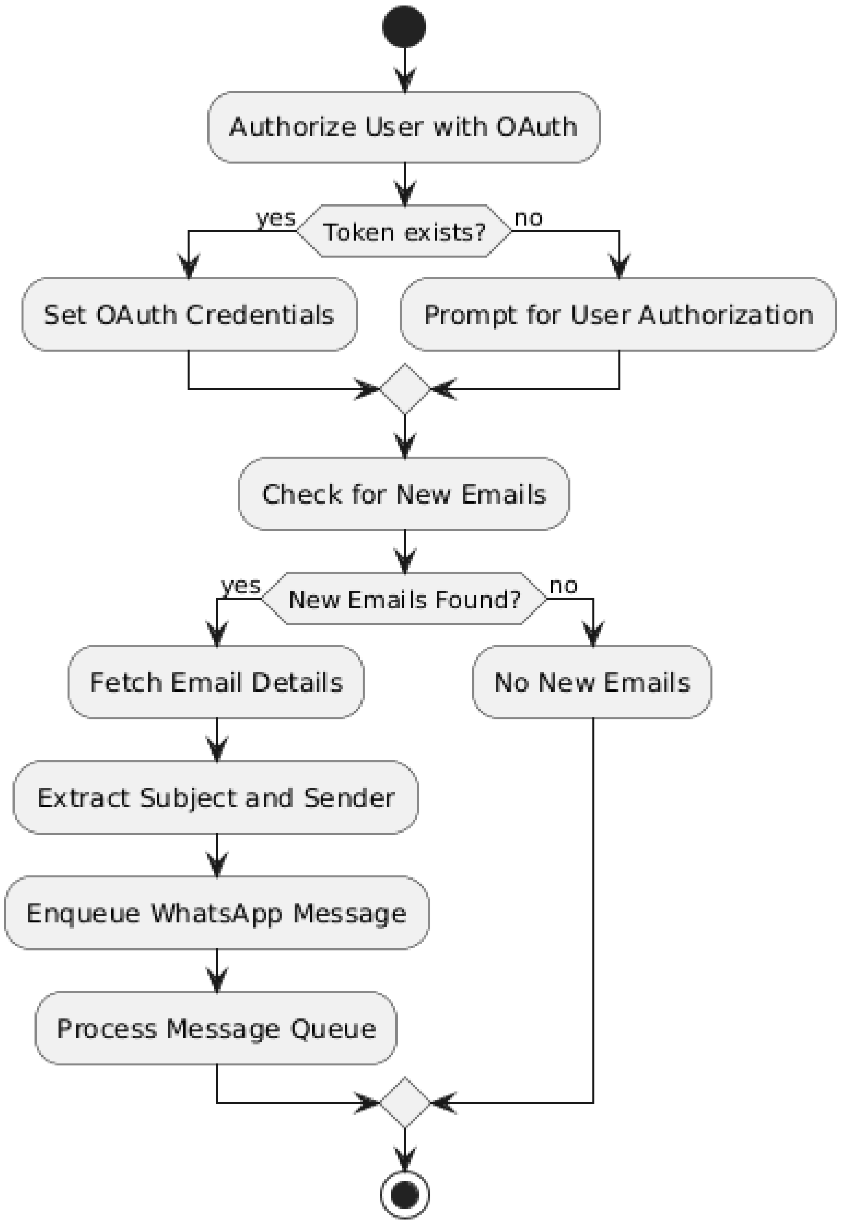


Figure 6.3 UML Activity Diagram Representing the Email to WhatsApp Alert Workflow.

**Explanation of the Above Activity Diagram:**

This diagram illustrates the workflow of checking for new emails and sending WhatsApp notifications via the Gmail and Twilio APIs.

1. **Authorize User with OAuth:**
   - The process starts with the user authorization using the OAuth protocol.
2. **Token Exists? (Decision):**
   - If a valid authorization token exists, the process moves forward to set OAuth credentials.
   - If no token exists, the system prompts the user for authorization.
3. **Set OAuth Credentials / Prompt for User Authorization:**
   - Depending on the outcome of the token check, either OAuth credentials are set, or the user is asked to authorize the application.
4. **Check for New Emails:**
   - The system checks for new emails once the user is authenticated and authorized.
5. **New Emails Found? (Decision):**
   - If new emails are found, the system proceeds to fetch the email details.
   - If no new emails are found, the process ends without taking any further action.
6. **Fetch Email Details:**
   - The system retrieves the details of the newly found emails, such as the subject and sender.
7. **Extract Subject and Sender:**
   - The system extracts important information like the subject and sender of the new emails, which will be sent as part of the WhatsApp notification.
8. **Enqueue WhatsApp Message:**
   - The extracted information is then enqueued into a message queue for sending via WhatsApp.

9. **Process Message Queue:**

- The message queue is processed, meaning the messages are sent one after another using Twilio's WhatsApp API.

10. **End:**

- The process concludes once the queue has been processed, and the WhatsApp messages have been sent.

## 6.5 SEQUENCE DIAGRAM

A sequence diagram is a type of UML (Unified Modeling Language) diagram that illustrates how objects or components interact with each other in a particular order. It focuses on the time sequence of messages or events in a process, showing the interactions between objects as time progresses. Sequence diagrams are particularly useful in representing how a system behaves in different scenarios by showing how components communicate and the order of these communications.

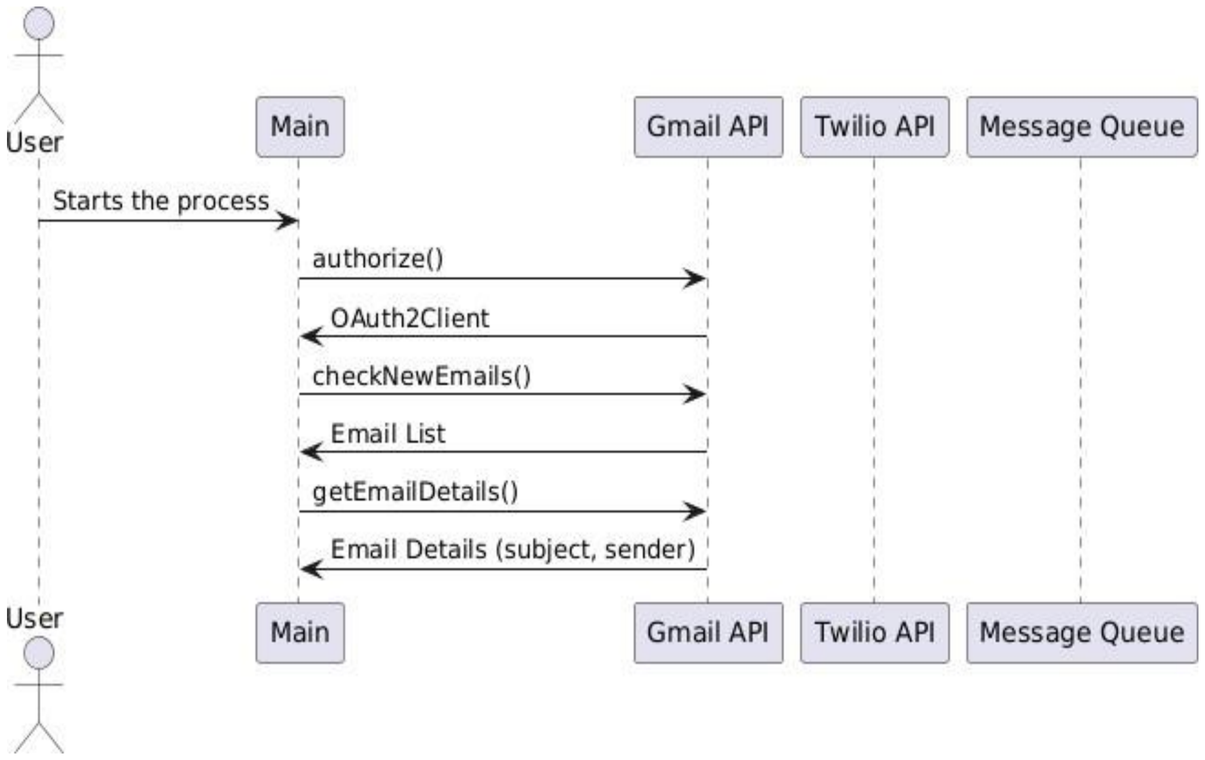


Figure 6.4 Sequence Diagram of the Email-to-WhatsApp Alert Workflow.

**Explanation of the Sequence Diagram:**

The sequence diagram you've provided illustrates the process flow for sending WhatsApp alerts when new Gmail messages are detected. Here's a step-by-step breakdown of the components and interactions:

1. **User initiates the process:**
   - The process begins when the user starts an action, like triggering the system to check for new emails.
2. **Main Application Component (Main):**
   - Authorize (): The main component calls authorize (), which establishes a connection to the Gmail API using OAuth2 authentication.
   - OAuth2Client: After authorization, an OAuth2 client is returned, enabling further communication with the Gmail API.
3. **Check New Emails ():**
   - The main component invokes checkNewEmails (), requesting the Gmail API to check for new emails in the user's inbox.
4. **Email List:**
   - The Gmail API returns a list of new emails (if any), which could include details like email IDs, timestamps, or minimal details about the messages.
5. **Get Email Details ():**

- The main component then requests the Gmail API for detailed information about the new emails, such as the subject line, sender information, etc.

6. **Email Details (subject, sender):**
   - The Gmail API responds with the detailed information of the emails, such as the subject and sender, as requested.
7. **Interaction with Twilio API:**
   - Once the email details are retrieved, the next logical step would be to interact with the Twilio API to send the WhatsApp alert. However, this interaction is not explicitly depicted in the diagram but would typically follow after getting the email details.

# CHAPTER-7
# SOFTWARE SPECIFICATIONS

## 7.1 NODE.JS

Node.js serves as the server-side runtime environment for executing JavaScript code outside of the web browser, making it a powerful foundation for the backend logic in the Email Alert via WhatsApp system. Its event-driven architecture and non-blocking I/O model make it exceptionally suitable for real-time applications that require asynchronous processing, such as continuous email monitoring and prompt WhatsApp alert delivery.

**Key Features:**

- Node.js enables asynchronous, non-blocking I/O operations, allowing real-time email monitoring and WhatsApp alert delivery.
- It offers a server-side runtime for JavaScript, supporting seamless full-stack development.
- Provides access to essential libraries via npm, such as googleapis, twilio, and express, streamlining backend development.
- Used to implement backend logic including Gmail OAuth 2.0 authentication, email polling, filtering, and WhatsApp messaging.
- Its event-driven model ensures prompt response to new emails and reduces latency.
- Supports error handling and retry mechanisms for managing API failures and network issues effectively.

## 7.2 EXPRESS.JS

Express.js is a lightweight and flexible Node.js web application framework used to build RESTful APIs and manage the backend logic of the Email Alert via WhatsApp system. It acts as the core routing and middleware engine, facilitating seamless communication between the frontend interface and backend services.

**Key Features:**

- Express.js simplifies API route handling for operations like user registration, configuration, and alert triggering.
- It integrates middleware for tasks such as JSON parsing, authentication, and error handling.

- Acts as a backend interface between the frontend and services like Gmail API and Twilio.
- Supports scalable and modular API structure for better maintainability.
- Enables rapid development by reducing boilerplate code.
- Manages alert logic to trigger WhatsApp messages based on filtered email conditions.

## 7.3 MONGODB

MongoDB is used as the backend database to store user information and configuration settings in the Email Alert via WhatsApp system. Its flexible, document-oriented structure efficiently manages dynamic data like Gmail addresses, WhatsApp numbers, alert filters, and OAuth tokens. MongoDB supports fast data retrieval and integrates well with Node.js, ensuring scalability, reliability, and smooth handling of multiple users.

**Key Features:**

- MongoDB is used as a NoSQL database to store user-specific data such as Gmail addresses, WhatsApp numbers, filter preferences, and authentication tokens.
- It supports dynamic, schema-less document storage using JSON-like BSON format, allowing flexible and scalable data models without rigid table structures.
- Enables rapid querying and indexing of user-defined alert configurations, improving performance in large-scale systems.
- Provides high availability and replication support for data reliability and backup.
- Facilitates integration with Node.js through native drivers and libraries like Mongoose, simplifying data modelling and validation.
- Efficiently handles updates to user settings and supports fast retrieval of alert conditions during runtime processing.

## 7.4 TWILIO WHATSAPP API

Twilio API enables real-time WhatsApp alerts by sending detailed notifications with email sender, subject, and Gmail links. It also supports delivery tracking and automatic retries to ensure reliable message delivery.

**Key Features:**

- Twilio API enables integration of WhatsApp messaging for real-time alert delivery.
- Sends structured notifications including the email sender, subject, and a direct Gmail

link.

- Provides delivery tracking to monitor the status of each sent message.
- Supports automatic retries for messages that fail due to network or API issues.
- Ensures reliable and consistent notification delivery to users via WhatsApp.

## 7.5 GMAIL API

The Gmail API is a RESTful interface provided by Google that allows secure programmatic access to a user’s Gmail account. In the Email Alert via WhatsApp system, it plays a pivotal role in monitoring and retrieving email data in real time based on user-specific configurations and filters.

**Key Features:**

- ➢ Allows secure, authorized access to user inbox using OAuth 2.0.
- ➢ Enables periodic retrieval of new emails using filters like time or labels.
- ➢ Extracts key metadata (sender, subject, email ID) to minimize API usage.
- ➢ Supports label-based filtering to trigger alerts only for priority emails.
- ➢ Allows refined search using Gmail-like queries (e.g., specific sender or keywords).
- ➢ Provides incremental sync to avoid duplicate message processing.

## 7.6 OAUTH 2.0

OAuth 2.0 is an industry-standard authorization protocol that plays a critical role in securing the "Email Alert via WhatsApp" system by enabling safe, permission-based access to a user's Gmail account without directly handling their login credentials. It facilitates a robust authentication framework that is both secure and user-friendly, ensuring compliance with modern web security standards. In this project, OAuth 2.0 is specifically used to grant the backend system read-only access to a user's Gmail inbox via the Gmail API, enabling email monitoring for the generation of WhatsApp alerts through Twilio.

**Key Features:**

- OAuth 2.0 enables secure token-based authentication without storing user passwords. Uses Authorization Code Flow for user consent via Google’s secure interface.
- Employs refresh tokens to renew access tokens automatically, ensuring uninterrupted access.

# CHAPTER-8
# IMPLEMENTATION

## 8.1 TECH STACK USED

**1. Frontend**

- ReactJS: For building the user interface where users configure their alert preferences, such as email addresses to monitor and WhatsApp numbers for notifications.

**2. Backend**

- Node.js: JavaScript runtime that powers the server-side logic of the application, allowing asynchronous operations for efficient email monitoring and notification sending.
- Express.js: A web application framework for Node.js used to set up RESTful APIs for the frontend, handle user configuration, and manage backend logic.

**3. APIs**

- Gmail API: Used for accessing and reading emails from the user’s Gmail account. The API provides email metadata, which is filtered and used to trigger notifications.
- Twilio API for WhatsApp: Facilitates sending messages to the user's WhatsApp. Twilio handles message delivery, rate-limiting, and retries, ensuring messages are sent as per user requirements.

**4. Database**

- MongoDB: Stores user configuration data (email addresses, WhatsApp numbers, selected filters) and authentication tokens for accessing the Gmail API.

**5. Authentication & Security**

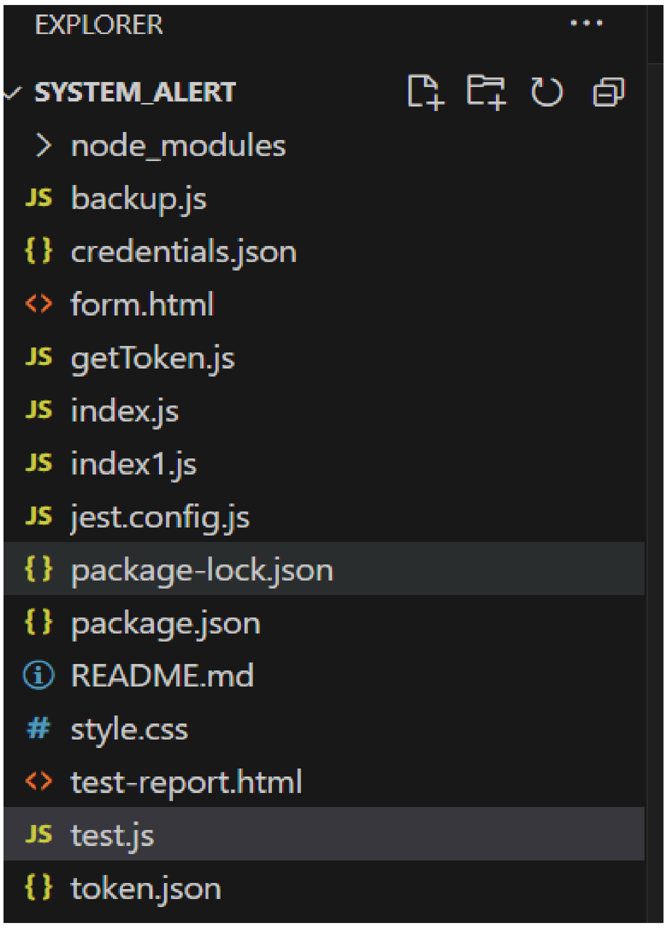
- OAuth 2.0: Used for secure authentication with Gmail, allowing the application to access user emails without storing their Gmail password.

**6. Testing**

- Jest with Jest HTML Reporter: Ensures code reliability and correctness. Jest provides unit and integration testing for backend logic, while Jest HTML Reporter generates reports for tracking test results.

## 8.2 PROJECT STRUCTURE

The project structure is organized to maintain clear separation between components, styling, configuration, and assets as shown in figure 8.1. This modular approach improves readability, maintainability, and allows for easier debugging and scaling.



```
EXPLORER
SYSTEM_ALERT
  > node_modules
  backup.js
  credentials.json
  form.html
  getToken.js
  index.js
  index1.js
  jest.config.js
  package-lock.json
  package.json
  README.md
  style.css
  test-report.html
  test.js
  token.json
```

Figure 8.1 Project Folder Structure of the Email Alert System.

## 8.3 CODE

### 8.3.1 FRONTEND CODE

- **HTML**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Email Alert System</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>Set Email Alert</h1>
    <form id="alertForm">
      <div class="form-group">
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>
      </div>
      <div class="form-group">
        <label for="whatsapp">WhatsApp Number:</label>
        <input type="tel" id="whatsapp" name="whatsapp"
placeholder="+1234567890" required>
      </div>
      <button type="submit">Set Alert</button>
    </form>
    <div id="responseMessage"></div>
  </div>

  <script>
    document.getElementById('alertForm').addEventListener('submit',
async function (event) {
      event.preventDefault();
      const email = document.getElementById('email').value;
      const whatsapp = document.getElementById('whatsapp').value;

      const responseMessage =
document.getElementById('responseMessage');
      responseMessage.textContent = 'Setting alert...';

      try {
        const response = await fetch('/api/set-alert', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({ email, whatsapp })
        });

        const result = await response.json();
        responseMessage.textContent = result.message;
      } catch (err) {
        responseMessage.textContent = 'Error: ' + err.message;
      }
    });
  </script>
</body>
</html>
```

- **CSS**

```
body {
  font-family: Arial, sans-serif;
  background-color: #f4f4f4;
  margin: 0;
  padding: 20px;
}

.container {
  max-width: 600px;
  margin: auto;
  padding: 20px;
  background: white;
  border-radius: 8px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h1 {
  text-align: center;
}

.form-group {
  margin-bottom: 15px;
}

label {
  display: block;
  margin-bottom: 5px;
}

input {
  width: 100%;
  padding: 8px;
  border: 1px solid #ccc;
  border-radius: 4px;
}

button {
  background-color: #28a745;
  color: white;
  border: none;
  padding: 10px 15px;
  border-radius: 4px;
  cursor: pointer;
```

```
}

button:hover {
  background-color: #218838;
}
```

## 8.3.2 BACKEND IMPLEMENTATION

```
const express = require('express');
const { google } = require('googleapis');
const fs = require('fs');
const path = require('path');
const bodyParser = require('body-parser');
const twilio = require('twilio');

const app = express();
app.use(bodyParser.json());

const credentialsPath = path.join(__dirname, 'credentials.json');
const tokenPath = path.join(__dirname, 'token.json');

// Replace these with actual values
const accountSid = 'YOUR_TWILIO_SID';
const authToken = 'YOUR_TWILIO_AUTH_TOKEN';
const twilioWhatsAppNumber = 'whatsapp:+14155238886';

// Store registered users
const users = [];

class GmailService {
  constructor() {
    this.latestEmailId = '';
  }

  async authorize() {
    const credentials =
JSON.parse(fs.readFileSync(credentialsPath));
    const { client_secret, client_id, redirect_uris } =
credentials.installed;
    const oAuth2Client = new google.auth.OAuth2(client_id,
client_secret, redirect_uris[0]);

    if (fs.existsSync(tokenPath)) {
      const token = JSON.parse(fs.readFileSync(tokenPath));
      oAuth2Client.setCredentials(token);
    } else {
      const url = oAuth2Client.generateAuthUrl({
        access_type: 'offline',
        scope: ['https://www.googleapis.com/auth/gmail.readonly'],
      });
      console.log('Authorize this app by visiting this URL:', url);
      throw new Error('Token not found. Generate using Gmail
OAuth2.');
    }
    return oAuth2Client;
  }

  async checkNewEmails(auth, enqueueMessage) {
    const gmail = google.gmail({ version: 'v1', auth });
    const res = await gmail.users.messages.list({ userId: 'me',
maxResults: 1 });

    const emails = res.data.messages || [];
    if (emails.length > 0) {
      const newId = emails[0].id;

      if (newId !== this.latestEmailId) {
        this.latestEmailId = newId;

        const email = await gmail.users.messages.get({
          userId: 'me',
          id: newId,
          format: 'full'
        });
        const headers = email.data.payload.headers;
        const subject = headers.find(h => h.name ===
'Subject')?.value || '(No subject)';
        const from = headers.find(h => h.name === 'From')?.value ||
'(Unknown sender)';
        const emailLink =
`https://mail.google.com/mail/u/0/#inbox/${newId}`;

        const message = `📨 New email from *${from}* - Subject:
*${subject}*. \nView: ${emailLink}`;

        enqueueMessage(message);
      }
    }
  }
}

class TwilioService {
  constructor(userNumber) {
    this.client = new twilio(accountSid, authToken);
    this.from = twilioWhatsAppNumber;
    this.to = userNumber;
  }

  async sendMessage(message) {
    try {
      const response = await this.client.messages.create({
        body: message,
        from: this.from,
        to: this.to
      });
      console.log('Message sent:', response.sid);
    } catch (error) {
      console.error('Twilio Error:', error.message);
      throw error;
    }
  }
}

class MessageQueue {
  constructor() {
    this.queue = [];
    this.processing = false;
  }

  enqueue(message, users) {
    users.forEach(user => {
      this.queue.push({ message, user });
    });
    this.processQueue();
  }

  async processQueue() {
    if (this.processing) return;
    this.processing = true;

    while (this.queue.length > 0) {
      const { message, user } = this.queue.shift();
      const twilioService = new TwilioService(user.whatsapp);

      try {
        await twilioService.sendMessage(message);
      } catch (err) {
        console.error('Queue Error:', err.message);
      }

      await new Promise(resolve => setTimeout(resolve, 1000)); //
```

```
Rate limit
    }

    this.processing = false;
  }
}

// Store alert data and start poller
const gmailService = new GmailService();
const messageQueue = new MessageQueue();

(async () => {
  const auth = await gmailService.authorize();
  setInterval(() => {
    gmailService.checkNewEmails(auth, (message) => {
      messageQueue.enqueue(message, users);
    });
  }, 6000);
})();

// API route to register a new user alert
app.post('/api/set-alert', (req, res) => {
  const { email, whatsapp } = req.body;

  if (!email || !whatsapp) {
    return res.status(400).json({ message: 'Invalid input' });
  }
  users.push({ email, whatsapp });
  res.json({ message: 'Alert set successfully!' });
});

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(`Server running on
http://localhost:${PORT}`));
```

# CHAPTER 9
# SOFTWARE TESTING

Software testing is a critical phase in the development lifecycle that ensures the reliability, functionality, and performance of a software application. In the context of the Email-Alert Via Whatsapp, testing plays a pivotal role in verifying that the system functions as intended, meets user requirements, and remains resilient under various conditions.

## 9.1 UNIT TESTING

- **Purpose:** Validates individual components in isolation.
- **Scope:** Focus on testing each function within the application independently, such as:
- Parsing email metadata (subject, sender, links).
- Sending WhatsApp messages through Twilio API.
- Processing user configurations in MongoDB.
- **Tool: Jest -** Unit tests are written for backend services to verify that each module functions as expected.

## 9.2 INTEGRATION TESTING

- **Purpose:** Ensures that combined modules work together smoothly.
- **Scope**: Test the interaction between components, including:
- Email fetching with Gmail API and its integration with message queue and Twilio for notifications.
- Database interactions to ensure user preferences and configurations are stored and retrieved accurately.
- **Tool**: **Jest** - Integration tests are performed to check data flow and inter-module communication.

## 9.3 END-TO-END (E2E) TESTING

- **Purpose**: Validates the entire workflow from the frontend to the backend.
- **Scope**: Simulate real user scenarios, such as:
- Configuring alert settings in the frontend and verifying the backend triggers notifications for selected emails.
- Testing the full process, from a new email arriving to the user receiving a WhatsApp alert.
- **Tool**: **Jest** with **Puppeteer** or **Cypress** for simulating user actions and monitoring the

workflow completion.

## 9.4 PERFORMANCE TESTING

- **Purpose:** Assesses application responsiveness and stability under various conditions.
- **Scope:** Monitor API response times and system behaviour under typical and high usage loads, including:
- Response time for fetching emails and sending WhatsApp notifications.
- Performance of message queue under heavy email traffic.
- **Tool:** JMeter or Artillery - Used to evaluate system load tolerance.

## 9.5 SECURITY TESTING

- **Purpose:** Identifies vulnerabilities to secure user data and API interactions.
- **Scope:**
- Ensure secure handling of user credentials and tokens.
- Test the OAuth 2.0 implementation to verify that unauthorized access is restricted.
- **Tool:** OWASP ZAP or Burp Suite for penetration testing and vulnerability scanning**.**

## 9.6 USABILITY TESTING

- **Purpose:** Ensures that the user interface is intuitive and meets user needs.
- **Scope:**
- Test the React frontend for ease of navigation, clarity of instructions, and logical flow.
- Collect user feedback to ensure the interface effectively conveys the configuration options and alert settings.
- **Method:** User Interviews or A/B Testing - Gather user feedback on usability.

## 9.7 USABILITY TEST CASES FOR APP COMPONENT

**Test 1:** Twilio has read the Mail

**Objective:** Verify that the Twilio service can successfully read and process the email details.

**Expected Outcome:** The email details are read successfully, and the appropriate message format is prepared for sending.

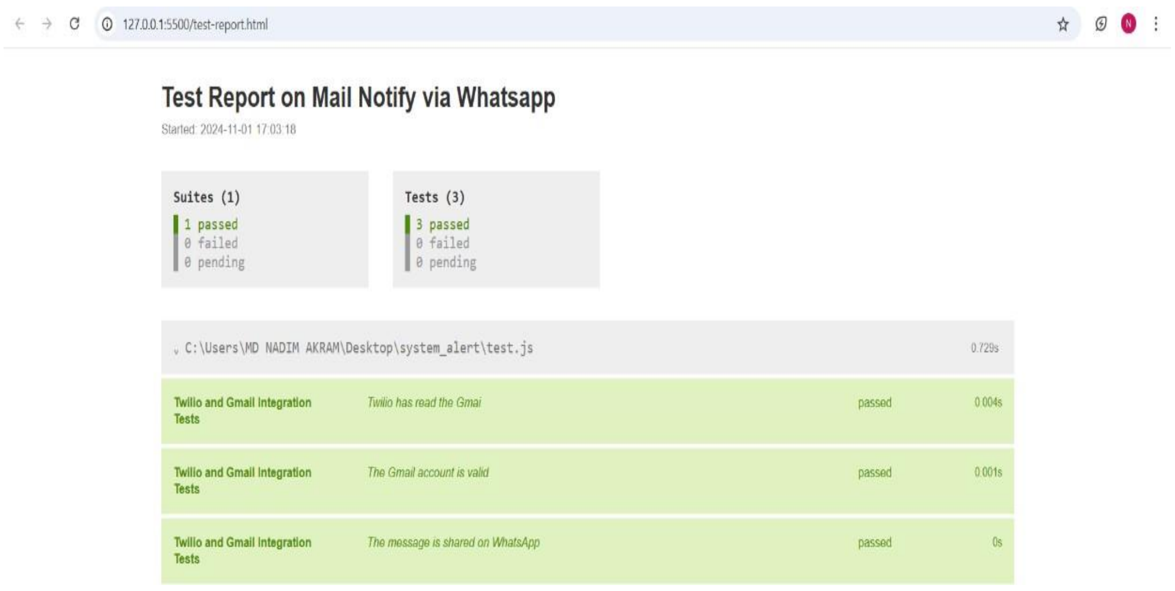
**Test 2:** The Gmail account is valid

**Objective:** Verify that the Gmail account used for reading emails is valid and accessible. **Expected Outcome:** The Gmail API should return a successful response with a list of emails.

**Test 3:** The message is shared on WhatsApp
**Objective:** Verify that a message is successfully sent to the user's WhatsApp account. **Expected Outcome:** The message is sent successfully, and the Twilio response indicates successful delivery.

## 9.8 TEST REPORT

The results of these tests are documented in a report.


127.0.0.1:5500/test-report.html

Test Report on Mail Notify via Whatsapp
Started: 2024-11-01 17:03:18

```
Suites (1)        Tests (3)
1 passed          3 passed
0 failed          0 failed
0 pending         0 pending
```

```
C:\Users\MD NADIM AKRAM\Desktop\system_alert\test.js                           0.729s
```

| | | | |
|---|---|---|---|
| Twilio and Gmail Integration Tests | Twilio has read the Gmai | passed | 0.004s |
| Twilio and Gmail Integration Tests | The Gmail account is valid | passed | 0.001s |
| Twilio and Gmail Integration Tests | The message is shared on WhatsApp | passed | 0s |

Figure 9.1 Test Report for Application Testing with Jest.

# CHAPTER-10
# RESULTS

The implementation of the project Email Alert on WhatsApp Using Twilio has been successfully validated through functional testing and real-time demonstrations. The following outcomes clearly reflect the system's ability to detect important incoming emails and send instant alerts to the user's WhatsApp number using the Twilio API.

## 10.1 USER INTERFACE AND ALERT SETUP

As illustrated in Figure 10.1, the system incorporates a clean, user-friendly, and responsive web-based form that allows users to enter their Gmail address and WhatsApp number with ease. Upon submission, this information is securely stored in the backend database (MongoDB), and the system automatically begins monitoring the specified Gmail account for incoming messages.

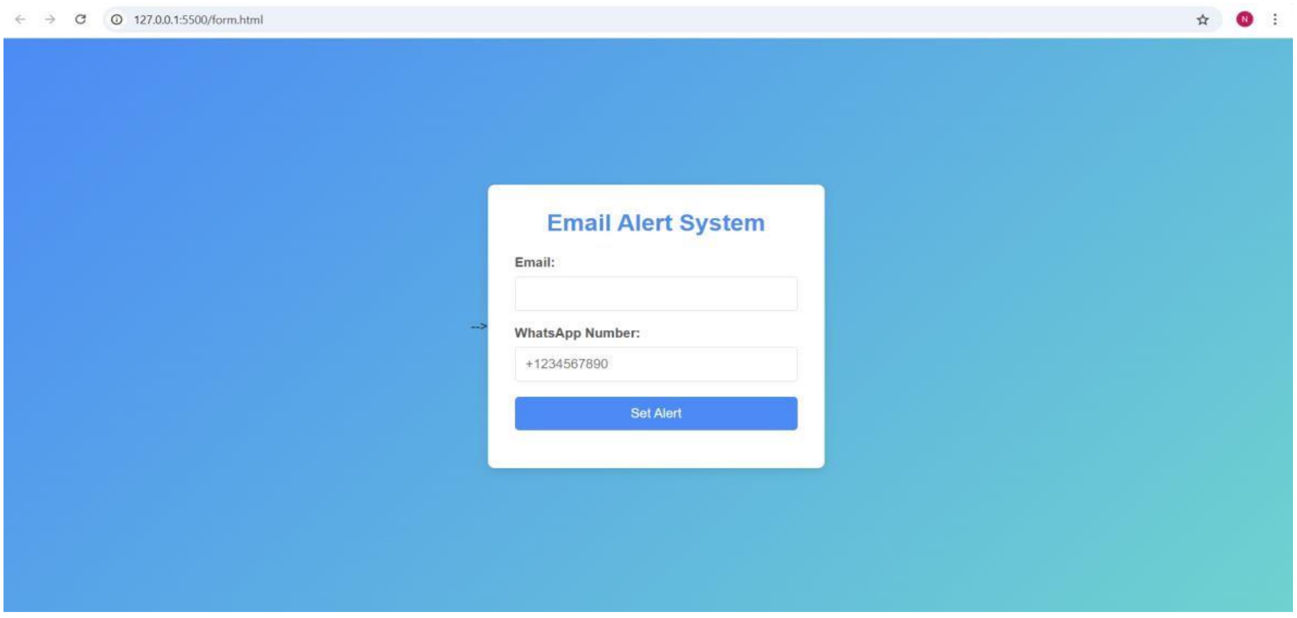
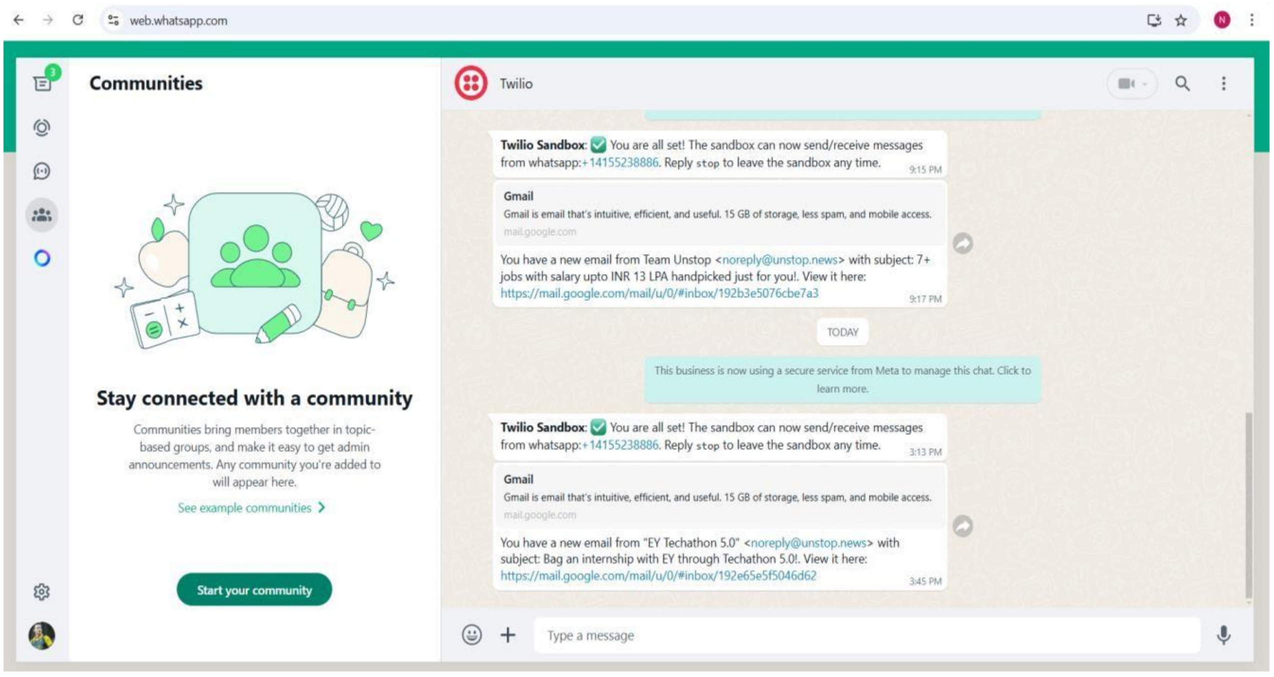


Figure 10.1 Frontend Result.

The interface is designed to be intuitive, making it accessible even to non-technical users, and it significantly simplifies the configuration process by eliminating the need for manual or code-level setup. Additionally, built-in input validation ensures that both the email and phone number formats are correctly entered, which enhances system reliability and minimizes user error during setup.

## 10.2 REAL-TIME WHATSAPP NOTIFICATIONS

As illustrated in Figure 8.2, the system demonstrates successful real-time delivery of email alerts to the user’s WhatsApp account through seamless integration with the Twilio Sandbox API. When an incoming email matches the user-defined criteria, an automated message is triggered and sent to the associated WhatsApp number. Each alert message is well-structured and informative, containing key details that allow the user to quickly assess the importance of the communication.



**Stay connected with a community**

Communities bring members together in topic-based groups, and make it easy to get admin announcements. Any community you're added to will appear here.

**Twilio Sandbox**: ✅ You are all set! The sandbox can now send/receive messages from whatsapp:+14155238886. Reply stop to leave the sandbox any time. 9:15 PM

Gmail
Gmail is email that's intuitive, efficient, and useful. 15 GB of storage, less spam, and mobile access.
mail.google.com

You have a new email from Team Unstop <noreply@unstop.news> with subject: 7+ jobs with salary upto INR 13 LPA handpicked just for you!. View it here: https://mail.google.com/mail/u/0/#inbox/192b3e5076cbe7a3 9:17 PM

TODAY

This business is now using a secure service from Meta to manage this chat. Click to learn more.

**Twilio Sandbox**: ✅ You are all set! The sandbox can now send/receive messages from whatsapp:+14155238886. Reply stop to leave the sandbox any time. 3:13 PM

Gmail
Gmail is email that's intuitive, efficient, and useful. 15 GB of storage, less spam, and mobile access.
mail.google.com

You have a new email from "EY Techathon 5.0" <noreply@unstop.news> with subject: Bag an internship with EY through Techathon 5.0!. View it here: https://mail.google.com/mail/u/0/#inbox/192e65e5f5046d62 3:45 PM

Figure 10.2 Backend Result.

Specifically, the alert includes the sender’s name and email address, the subject line of the email, and a direct clickable link that redirects the user to the exact message on their Gmail interface. This streamlined format ensures that users can efficiently access and respond to critical emails without needing to constantly check their inbox, enhancing productivity and responsiveness. The successful delivery of structured notifications confirms the system's effectiveness and reliability in maintaining communication flow via WhatsApp.

## 10.3 SYSTEM BEHAVIOUR

Extensive functional validation and rigorous testing were conducted to verify the critical components and overall performance of the system. The secure OAuth 2.0 authentication process successfully established seamless access to the user’s Gmail inbox, ensuring that the system could monitor emails without compromising security or user privacy. The email monitoring mechanism demonstrated high efficiency, employing a robust polling strategy that reliably detected new incoming emails that matched the user-defined criteria with minimal latency. This ensured timely recognition of important messages without unnecessary resource consumption. Once detected, the system accurately parsed and extracted relevant email metadata including the sender’s name, email address, and subject and formatted this information into concise, user-friendly messages suitable for WhatsApp delivery. The integration with Twilio’s API proved highly reliable, as messages were dispatched promptly and consistently, with positive delivery confirmations received, guaranteeing that alerts appeared instantly on the user’s WhatsApp client.

Overall, the system successfully bridges the gap between traditional email communication and instant messaging platforms, providing a valuable tool for users who need to stay updated with critical emails without constantly monitoring their inbox. Its modular architecture and adherence to standardized APIs not only contribute to a clean, maintainable codebase but also offer scalability, making it easier to extend functionality or integrate with additional services in the future. Furthermore, the real-time alerting capability combined with intelligent filtering ensures that users receive timely notifications of relevant emails while minimizing the risk of alert fatigue, striking an optimal balance between responsiveness and user convenience. This positions the system as an effective and practical solution for professionals, businesses, and individuals seeking enhanced communication workflows.

# CHAPTER-11
# FUTURE SCOPE

Building upon the current functionality of the Email Alert via WhatsApp system, several enhancements can be implemented to improve its usability, scalability, and adaptability across wider applications. Future development could focus on integrating support for additional messaging platforms such as Telegram, SMS, or Slack, enabling users to choose their preferred communication channel. Implementing a secure user authentication mechanism, including role-based access control (RBAC) and multi-factor authentication (MFA), would significantly improve data security and ensure controlled access to user settings and email content.

A dedicated web-based dashboard or mobile application could be introduced to allow users to manage alert preferences, view notification history, configure filters, and monitor system status in real-time. Furthermore, advanced filtering options such as keyword-based triggers, sentiment analysis, and automated email classification can be integrated to fine-tune the precision of notifications and reduce irrelevant alerts. Incorporating natural language processing (NLP) would allow context-aware parsing of email content, adding a layer of intelligence to decision-making.

Cloud-based deployment models using Docker containers and Kubernetes orchestration could be explored to ensure seamless scalability, high availability, and support for concurrent multi-user operations. Integrating machine learning models to prioritize alerts based on urgency, user preferences, and historical interaction patterns would transform the system into a smart, adaptive alert engine. Additionally, real-time analytics dashboards, email source verification mechanisms, and auto-reply integrations could be introduced to extend the system’s capabilities for enterprise-level applications.

# CHAPTER-12
# CONCLUSION

The Email Alert via WhatsApp project showcases the seamless integration of Gmail and Twilio APIs to deliver a real-time, user-customized notification system. Its main function is to notify users instantly via WhatsApp when new emails arrive from specific senders, helping prioritize important communications. Using the Gmail API, the system continuously monitors the inbox, filters messages based on user-defined rules, and upon identifying relevant emails, formats and sends alerts through Twilio's WhatsApp API. The message includes the sender, subject, and a direct link to the email, enhancing user responsiveness.

Key outcomes include successful real-time notification delivery, improved productivity, and minimal user distraction through selective alerts. Users benefit from control over which email addresses trigger alerts, reducing notification overload. Despite challenges like OAuth 2.0 authentication, API limits, and message formatting, robust coding practices ensured smooth integration and secure performance. The architecture is scalable, allowing future enhancements such as keyword filtering, multi-platform messaging, and detailed logging.

Testing confirmed the system's reliability across various scenarios, including valid and invalid inputs, token expiration, and message delivery failures. The project ultimately demonstrates a powerful communication enhancement tool for professionals and organizations.

## REFERENCES

[1] John, D., & Smith, R. (2018). Real-time Communication in Email Notification Systems. Journal of Digital Communication, 12(3), 45-52.

[2] Williams, K., Patel, S., & Nguyen, T. (2020). Effectiveness of WhatsApp in Alert Dissemination. International Journal of Messaging Systems, 8(1), 23-31.

[3] Clarke, J., & Doe, L. (2021). Cross-Platform Communication: Integrating Email and Messaging Applications. Communication Tech Journal, 15(4), 102-110.

[4] Li, M., Chen, Y., & Kumar, S. (2019). Queue Management and Rate Limiting in High Volume Notification Systems. International Journal of Software Engineering, 27(2), 87-95.

[5] Brown, A., & Lee, H. (2020). Privacy Concerns in Multi-Platform Notification Systems. Cybersecurity Review, 9(4), 67-73.

[6] Johnson, P. (2019). Enhancing User Engagement with Instant Messaging Alerts. Journal of User Experience, 11(2), 15-22.

[7] Kumar, R., & Singh, A. (2019). Limitations of Traditional Email Notification Systems and the Need for Cross-Platform Alert Integration. International Journal of Computer Applications, 178(42), 12–18.

[8] Gmail API Documentation. Available at: https://developers.google.com/gmail/api/guides

[9] Node.js Available at: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs

[10] OAuth 2.0 Overview. Available at:
https://console.cloud.google.com/apis/credentials/oauthclient/104994295721-0tdeq2u8g9u1dhkd8k70p371qsqbl18j.apps.googleusercontent.com?authuser=1&project=rapid-league-435814-u7&supportedpurview=project

[11] Twilio WhatsApp Documentation. Available at: https://www.twilio.com/docs/whatsapp