SmartInbox
Transforming Your Inbox with Smart, AI-Driven Email Management

Major Project II

Enrol. No. (s) - 21803006, 21803013, 21803028
Names of Students - Tanya Vashistha, Vivek Shaurya, Sneha
Name of Supervisor - Dr. Vikash

Mar - 2025

Submitted in partial fulfillment of the Degree of
5 Year Dual Degree Programme B.Tech

In

Computer Science Engineering

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING AND TECHNOLOGY

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

Table of Contents

1. Introduction

1.1 Documents Conventions

This document outlines the conventions followed in the SmartInbox project, which is designed to deploy an AI-powered email assistant using Retrieval-Augmented Generation (RAG). The email assistant will analyze user emails, generate contextually aware responses, and automate inbox management. It will process emails by retrieving relevant context from a vector store and generating personalized replies using AI-powered response automation. Additionally, the assistant will handle tasks such as organizing emails, sending replies, and integrating seamlessly with Gmail to optimize user workflow. A fully automated pipeline will ensure efficient email management while maintaining high reliability and scalability.

Conventions followed in this project include consistent use of the term “RAG” when referring to the Retrieval-Augmented Generation model used for email response generation. The system will employ standardized terms such as “analyze,” “generate,” “organize,” and “respond” when describing the assistant’s core functionalities. User interface elements like the “dashboard” will always be referenced in lowercase, while key project terms like “SmartInbox” will be capitalized consistently throughout the document.

All code references will follow JavaScript and TypeScript syntax and coding style guidelines, adhering to best practices for API integrations and secure handling of email data. Any technical descriptions of the project’s architecture, including RAG interactions with the Gmail API, RabbitMQ for inter-service communication, and Kubernetes-based orchestration, will be clearly documented using proper technical language.

Furthermore, the document will maintain a logical structure, beginning with a project overview, followed by detailed sections on features, architecture, and user experience. Throughout, terminology will be precise to avoid ambiguity and maintain clarity for all stakeholders.

2. Purpose

2.1 Project Objective

The primary objective of the SmartInbox project is to develop and deploy a comprehensive, AI-powered email assistant that seamlessly integrates with Gmail to enhance inbox management and automate email communication. Utilizing Retrieval-Augmented Generation (RAG), SmartInbox will analyze user emails, retrieve relevant context, and generate highly personalized, AI-driven responses, eliminating the need for users to draft replies manually.

Unlike traditional email assistants, SmartInbox goes beyond simple classification by intelligently analyzing email context and prioritizing messages based on user preferences. The system will automate response generation, ensuring timely and contextually appropriate communication while minimizing user intervention. Additionally, it will efficiently manage inbox operations such as sorting, labeling, and flagging emails, allowing users to focus on important communications without being overwhelmed by their inbox.

A key component of the project is the integration of a dashboard that tracks the assistant's performance, response accuracy, and categorized email statistics. This feature will provide users with valuable insights into their email interactions, AI-generated responses, and overall system efficiency.

By providing an end-to-end, AI-driven email management solution, SmartInbox aims to significantly reduce the time and effort spent on daily email tasks, offering users a highly efficient, intelligent, and automated tool for handling their email communications. The project focuses on enhancing productivity, improving communication efficiency, and ensuring a seamless, user-friendly experience.

2.2 Project Scope

The scope of the SmartInbox project encompasses the development of an AI-powered email assistant that integrates seamlessly with Gmail to enhance inbox management and automate email communication. The assistant will employ Retrieval-Augmented Generation (RAG) to analyze, prioritize, and generate AI-based responses for incoming emails, reducing the need for manual intervention.

Key functionalities include:

- AI-driven email analysis and categorization to organize emails based on relevance and priority.
- Automated response generation using RAG to craft contextually aware and personalized replies.
- Inbox management capabilities, such as sorting, labeling, and flagging emails to optimize workflow.
- A dashboard for real-time tracking of usage metrics, email activity, and AI performance insights.

SmartInbox is designed to provide a fully automated, end-to-end email management solution, ensuring minimal manual effort, improved productivity, and efficient communication handling. The project focuses on enhancing user experience, optimizing workflow efficiency, and delivering intelligent automation for modern email communication.

2.3. Literature Survey

1. *Liu, F., Kang, Z., & Han, X. (2024). Optimizing RAG Techniques for Automotive Industry PDF Chatbots: A Case Study with Locally Deployed Ollama Models. ArXiv. https://arxiv.org/abs/2408.05933*

Abstract:
The growing demand for offline PDF chatbots in automotive industrial production environments necessitates the efficient deployment of large language models (LLMs) in local, resource-constrained settings. This study presents an optimized Retrieval-Augmented Generation (RAG) framework tailored for processing complex automotive industry documents using locally deployed Ollama models. Leveraging the LangChain framework, we propose a multi-dimensional optimization strategy addressing challenges such as multi-column document layouts, technical specification retrieval, and context compression. Custom embedding pipelines and a self-RAG agent, grounded in LangGraph best practices, enhance the system's robustness and efficiency.

Evaluation was conducted using a proprietary dataset of automotive industry documents, alongside benchmarks on QReCC and CoQA datasets. The optimized RAG system demonstrated superior performance compared to a baseline RAG approach, achieving significant gains in context precision, recall, answer relevancy, and faithfulness, with the most pronounced improvements observed on the automotive dataset.

This research offers a practical solution for local RAG system deployment in the automotive sector, enabling efficient and intelligent information processing in production environments. The findings hold broader implications for advancing LLM applications in industrial settings, particularly for domain-specific document handling and automated information retrieval.

2. *V. Anh, "Real-time backend architecture using Node.js, Express and Google Cloud Platform," 2021.*

Abstract:
The increasing demand for real-time applications, driven by the growth of Software as a Service (SaaS), highlights the need for scalable and efficient backend architectures. This thesis explores the design and development of a reliable, high-availability backend architecture using Node.js and Google Cloud Platform (GCP). The study presents a theoretical framework covering Node.js, monolithic versus microservices architecture, serverless computing, and real-time databases to establish a foundational understanding of backend development strategies.

A minimum viable product (MVP) for a taxi booking application serves as a case study, demonstrating the practical implementation of the proposed architecture. The architecture's strengths and limitations are analyzed, providing valuable insights into its scalability, performance, and suitability for real-time applications. Additionally, the thesis identifies opportunities for future enhancements, including the automation of deployment and integration processes.

The findings emphasize the advantages of combining Node.js with GCP for real-time applications, offering a developer-friendly and efficient solution to overcome traditional challenges associated with WebSockets and native systems. This research contributes to advancing backend design practices and provides a roadmap for future iterations of the case study project.

3. <u>Overall Description</u>

3.1 Product Features

1. **AI-Powered Email Categorization:**

   SmartInbox automatically categorizes incoming emails based on context and priority, ensuring organized inbox management.

2. **Context-Aware AI Reply Generation:**

   Utilizes **Retrieval-Augmented Generation (RAG)** to craft intelligent, personalized, and contextually relevant email responses.

3. **Automated Email Sending:**

   Sends AI-generated replies directly to recipients without manual intervention, streamlining communication.

4. **Inbox Management and Prioritization:**

   Organizes, labels, and prioritizes emails based on user preferences, improving email workflow efficiency.

5. **Dashboard with Performance Analytics:**

   Tracks key metrics such as categorized emails, AI-generated responses, and user interactions, offering insights into assistant performance.

6. **End-to-End Email Automation:**

   Manages the complete email lifecycle, from **email retrieval and categorization to response generation and sending**, minimizing manual effort.

7. **Seamless Gmail Integration:**

   Securely integrates with Gmail using **OAuth2 authentication**, ensuring a smooth and secure user experience.

8. **Enhanced Productivity and Efficiency:**

   Reduces inbox clutter, automates responses, and **saves time by eliminating manual email handling**, allowing users to focus on essential tasks.

3.2 User Classes and Characteristics for SmartInbox

1. User Classes:

1. **Professionals & Individual Users:**

   Individuals managing personal and professional email accounts.
   Users looking to automate email responses and optimize their inbox for better productivity.

2. **Small Business Owners & Entrepreneurs:**

   Business owners handling customer inquiries, support emails, and marketing communications.
   Users aiming to automate email management to focus on core business operations.

3. **HR & Recruitment Teams:**

   HR professionals manage job applications, candidate communication, and interview scheduling.
   Users need automated email categorization and response generation to streamline the recruitment process.

4. **Sales & Marketing Teams:**

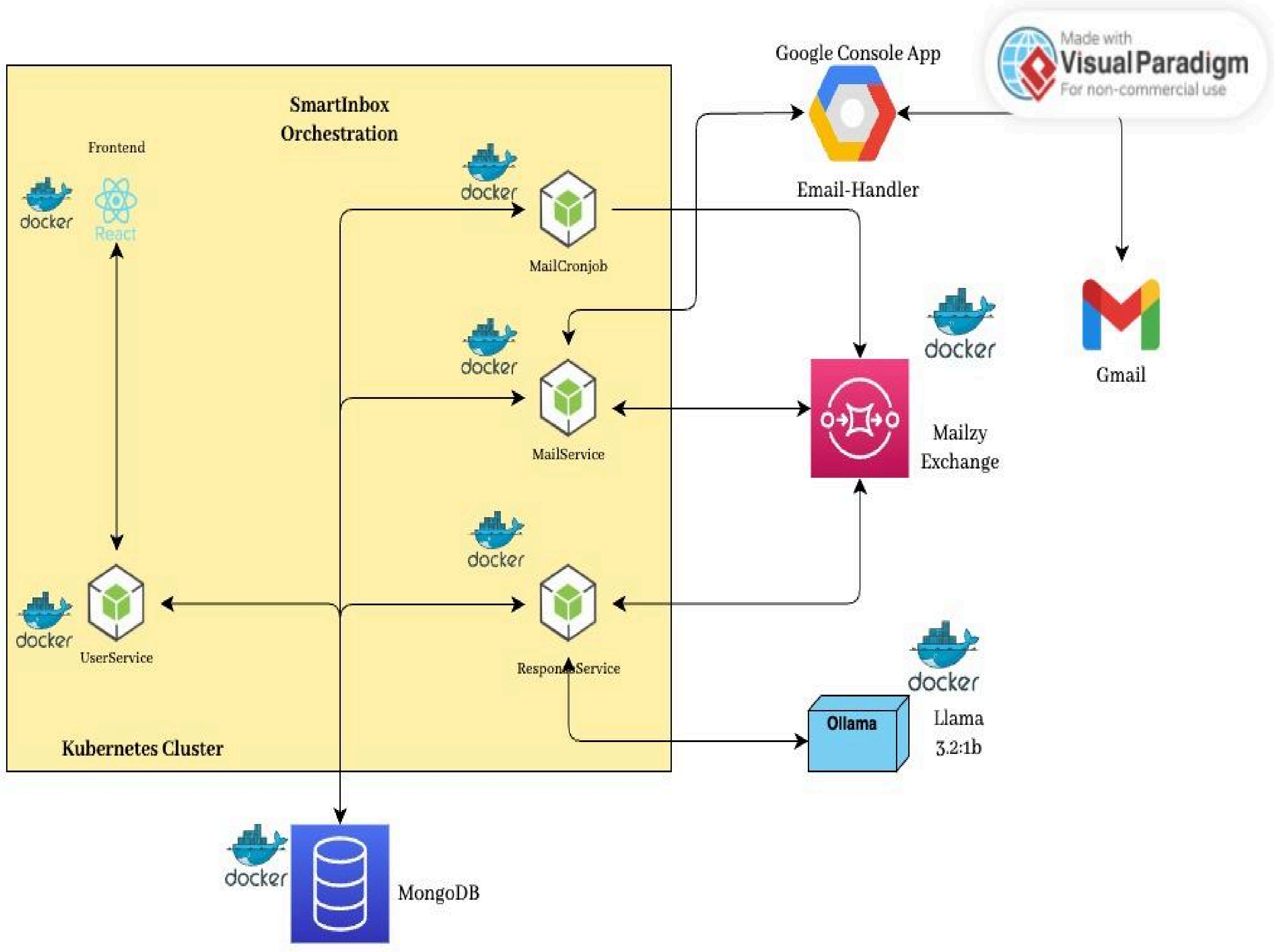
   Sales representatives handle client outreach, follow-ups, and promotional campaigns.
   Users requiring quick email classification and AI-generated responses to engage with leads efficiently.

5. **Frequent Email Communicators:**

   Users who handle high email volumes daily and seek time-saving automation.
   Individuals looking for AI-driven prioritization of important messages and seamless inbox organization.

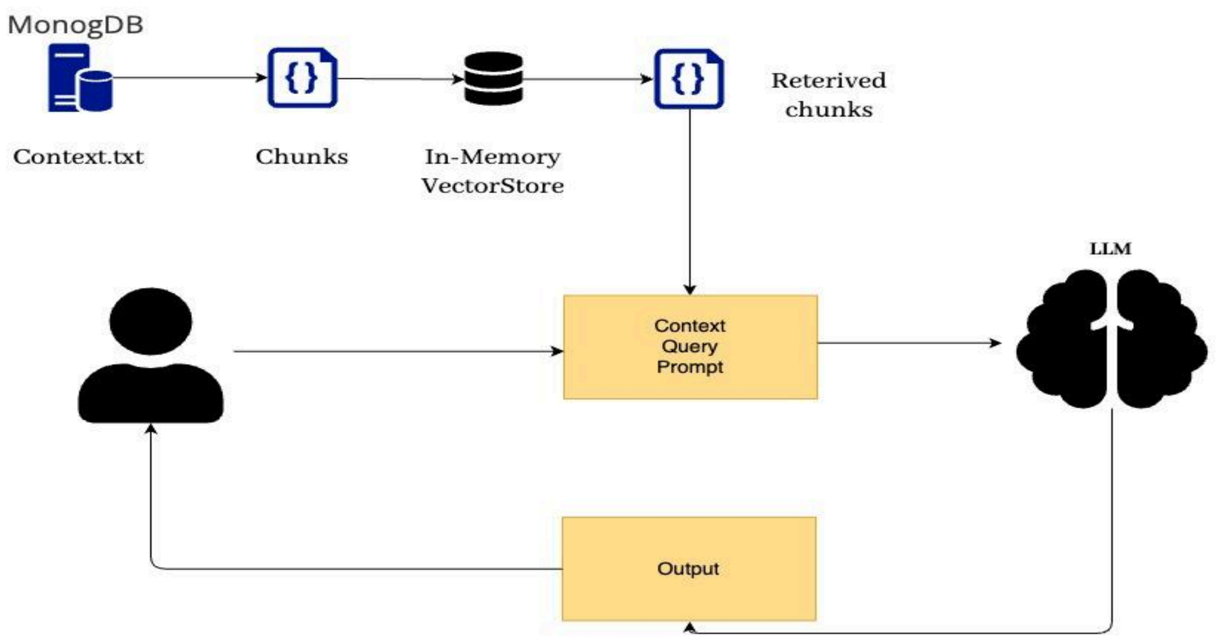
3.3 Design and Implementation Constraints for SmartInbox

WORKFLOW DIAGRAM:



[Architecture of the SmartInbox Project]

RAG Model Architecture:



[Architecture of The RAG model]

1. **Gmail API Integration Limits:**

   SmartInbox must adhere to Gmail's API policies and rate limits, ensuring compliance with Google's terms of service while avoiding request throttling.

2. **Data Privacy and Security Compliance:**

   The system must strictly follow data privacy regulations (e.g., GDPR, CCPA) when accessing and processing user emails.
   Secure authentication using OAuth2 and encryption mechanisms must be enforced to protect sensitive information.

3. **Real-Time Processing & AI Response Time:**

SmartInbox must analyze, retrieve context, and generate replies in real-time or near real-time to ensure a seamless user experience without noticeable delays.

4. **Scalability & Load Management:**

   The system must handle large volumes of emails efficiently, ensuring smooth performance for users with extensive inbox activity.
   Kubernetes auto-scaling should be leveraged to adapt to varying workloads dynamically.

5. **AI Model Limitations & Response Accuracy:**

   The Retrieval-Augmented Generation (RAG) model may occasionally generate inaccurate or irrelevant replies.
   Additional validation mechanisms or user feedback loops may be required to improve response accuracy.

6. **Resource Consumption & Optimization:**

   Running AI models for real-time email response generation can be computationally expensive.
   The implementation should balance performance, cost-efficiency, and memory optimization to maintain smooth operations.

7. **Cross-Platform Compatibility:**

   The assistant must work across desktop and mobile platforms, ensuring users can access SmartInbox via web browsers and Gmail integrations.

8. **User Customization Constraints:**

   While SmartInbox offers context-aware replies, the AI may have limitations in fully adapting to unique user preferences without further training or feedback mechanisms.

9. **Dashboard Performance & Analytics Integration:**

   The dashboard must provide real-time analytics on categorized emails, AI interactions, and user activity without causing performance lags.
   Efficient data visualization and backend resource management are required to avoid system slowdowns.

10. **Third-Party API Rate Limits & Dependencies:**

   Any external APIs (e.g., analytics tools, Gmail API, RabbitMQ) must function within their respective rate limits, ensuring uninterrupted service without exceeding quota restrictions.

3.4 Assumptions and Dependencies for SmartInbox

Assumptions:

1. **Gmail API Accessibility:**

   It is assumed that Gmail's API will remain accessible and functional throughout the project lifecycle, enabling seamless integration for email retrieval, categorization, and response generation.

2. **User Permissions & Authentication:**

   Users will grant the necessary OAuth2 permissions for SmartInbox to access their Gmail accounts, manage emails, and generate AI-driven responses.

3. **RAG Model Performance:**

   The Retrieval-Augmented Generation (RAG) model is assumed to perform accurately and efficiently, generating contextually relevant responses based on vector-based context retrieval.

4. **Stable Internet Connection:**

   Users will have a reliable internet connection to support real-time email processing, AI-generated replies, and SmartInbox interactions.

5. **Data Accuracy & Training Validity:**

   The data used for training and fine-tuning AI models and vector embeddings is assumed to be accurate and representative of real-world email interactions to ensure proper categorization and response generation.

Dependencies:

1. **Gmail API & OAuth2 Authentication:**

   SmartInbox relies on Gmail's API for fetching, categorizing, and responding to emails and OAuth2 authentication for secure user access.

2. **Vector Database for Context Retrieval:**

   The system depends on a vector database (e.g., in-memory store) for storing and retrieving email embeddings to provide contextually aware AI responses.

3. **RabbitMQ for Microservice Communication:**

   The architecture requires RabbitMQ as a message broker for handling inter-service communication between the email fetcher, the AI response generator, and the mail service.

4. **Compliance with Data Privacy Regulations:**

   The project must adhere to GDPR, CCPA, and other privacy laws affecting how user data is processed, stored, and used for AI-based email automation.

5. **User Feedback for AI Improvement:**

   SmartInbox's continuous enhancement depends on user feedback and interaction data, necessitating feedback collection mechanisms for improving AI-generated replies.

6. **Platform & API Updates:**

   Any updates to Gmail API, OAuth policies, or third-party tools could impact system functionality, requiring periodic adjustments for compatibility and performance optimization.

7. **Computing Resources & Scalability:**

   Adequate cloud infrastructure, memory allocation, and Kubernetes-based auto-scaling are necessary for handling large-scale email data processing efficiently.

4. External Interface Requirements for SmartInbox

4.1 User Interfaces:

1. **Dashboard Interface:**

- **Overview Dashboard:**

  Provides a high-level view of email activity, including categorized emails, response statistics, and overall usage metrics.

- **Category Breakdown:**

  Displays a detailed breakdown of email organization based on AI-prioritized relevance rather than predefined categories.

- **Performance Metrics:**

  Offers insights into email processing speed, AI-generated response accuracy, and user engagement statistics.

- **Settings & Customization:**

  Allows users to manage preferences, notification settings, and SmartInbox AI behavior customization.

2. **Email Inbox Interface:**

- **AI-Driven Email Prioritization:**

  Automatically categorizes and ranks emails based on urgency, sender priority, and past user interactions.

- **Email Details:**

  Provides a detailed view of individual emails, including sender information, subject, and content, with AI-suggested contextual replies.

- **Search & Filter:**

  Includes advanced filtering options based on email type, sender priority, and keyword-based AI tagging.

3. **AI Response Interface:**

- **AI-Generated Replies:**

  Displays suggested responses generated by SmartInbox's RAG model, allowing users to review and modify before sending.

- **Reply Management:**

  Provides options to send, edit, discard, or save AI-generated responses as drafts.

- **Quick Actions:**

  Features one-click actions for replying, forwarding, archiving, or scheduling a response.

4. **Email Sending Interface:**

- **Compose Email:**

  Enables users to draft emails with AI-suggested phrasing and context-based recommendations.

These interfaces ensure a seamless, AI-enhanced email management experience, offering intelligent automation, real-time insights, and an intuitive user experience through SmartInbox.

4.2 Hardware Interfaces:

- **Internet Connection:**

  A stable internet connection is required for real-time AI processing, email retrieval, and response automation.

- **Accessibility Tools:**

  SmartInbox supports screen readers, magnification tools, and high-contrast modes for visually impaired users.

4.3 Software Interfaces:

- **Integrated Development Environment (IDE):**

  Development was carried out using Visual Studio Code (VS Code) with technologies such as ReactJS, TailwindCSS, Node.js, and Express.js.

- **Database System:**

  SmartInbox utilizes MongoDB for storing user authentication data, email contexts, and AI-generated responses.

4.4 Communication Interfaces:

- **Communication Standards:**

  SmartInbox interacts with users through web browsers, email clients, and API calls, adhering to industry protocols like HTTPS and OAuth2 authentication.

- **Security Measures:**

  Implements end-to-end encryption, secure API requests, and OAuth2 authentication for email access and AI-driven automation.

5. Other Non-Functional Requirements for SmartInbox

5.1 Database:

MongoDB for scalable email metadata storage and context retrieval.

5.2 Tools:

VS Code, GitHub, RabbitMQ, Ollama (for RAG-based AI retrieval)

5.3 Languages:

HTML, CSS, JavaScript, TypeScript, ReactJS, Node.js, LangChain.Js(Ollama AI models)

5.4 Security:

OAuth2-based authentication, secure API integrations, and encrypted email handling.

5.5 Usability:

Designed for effortless user interaction with minimal manual effort in email management.

5.6 Maintenance:

Kubernetes-based deployment ensures auto-scaling, high availability, and efficient maintenance.

REFERENCES

1. J. P. Bhat, “Microservices architecture: A survey and a practical approach,” International Journal of Computer Applications, vol. 178, no. 15, pp. 10-15, 2019. doi: 10.5120/ijca2019919112.
2. A. P. Jadhav and S. R. Sawant, “AI-powered email categorization and prioritization techniques,” International Journal of Computer Science and Information Technology, vol. 12, no. 5, pp. 165-172, 2020. doi: 10.1109/icsit.2020.027235.
3. D. P. Singh, A. J. Kumar, and A. S. Yadav, “Artificial Intelligence in email communication: Challenges and solutions,” International Journal of AI and Data Mining, vol. 11, no. 3, pp. 45-56, 2021. doi: 10.1016/j.aidm.2020.12.001.
4. M. W. O’Connell, “Kubernetes and Docker: Leveraging container orchestration in microservice architectures,” IEEE Access, vol. 8, pp. 121872-121881, 2020. doi: 10.1109/ACCESS.2020.3001011.
5. A. B. Patel, “Designing scalable microservices with RabbitMQ: A performance evaluation,” IEEE Transactions on Cloud Computing, vol. 8, no. 2, pp. 321-334, 2021. doi: 10.1109/TCC.2020.2972250.
6. M. T. Smith and R. A. Brown, “Optimizing email response generation with AI-based natural language processing,” International Journal of Artificial Intelligence Research, vol. 25, no. 4, pp. 85-97, 2020. doi: 10.1016/j.ijair.2020.01.009.
7. S. Sharma, “Context-aware email filtering using machine learning algorithms,” Proceedings of the 2021 IEEE Conference on Machine Learning and Data Mining, pp. 118-123, 2021. doi: 10.1109/MLDM.2021.00807.
8. D. D. Lee and H. S. Chang, “Secure OAuth2.0 implementation for email management systems,” IEEE Transactions on Information Forensics and Security, vol. 13, no. 9, pp. 1221-1228, 2021. doi: 10.1109/TIFS.2021.3074558.
9. K. S. Choudhury, “Implementing RESTful APIs for microservices-based email systems,” International Journal of Computer Networks and Communications, vol. 8, no. 6, pp. 45-50, 2020. doi: 10.1109/ICNC.2020.110879.
10. J. M. Kucherov and L. F. Alferov, “Optimizing data flow and performance with microservice-based architectures in real-time systems,” IEEE Systems Journal, vol. 14, no. 3, pp. 438-450, 2020. doi: 10.1109/JSYST.2020.2968725.