

S-Mail

Minor Project-II

(ENSI252)

Submitted in partial fulfilment of the requirement of the degree of

BACHELOR OF TECHNOLOGY

to

K.R Mangalam University

by

Krish Punia (2301010136)

Kartikay Gautam (2301010137)

Gurnoor Kaur (2301010138)

Tanvi Basist (2301010166)

Under the supervision of

Dr. Amar Saraswat
Internal Mentor

Dr. Piyush Kumar
External Mentor
CEO
Indus Group Co.



Department of Computer Science and Engineering

School of Engineering and Technology

K.R Mangalam University, Gurugram- 122001, India

April 2025

CERTIFICATE

This is to certify that the Project Synopsis entitled, "**SMART CCTV**" submitted by "**Krish Punia (2301010137), Kartikay Gautam (2301010137) , Gurnoor Kaur (2301010138), Tanvi (2301010166)**" to **K.R Mangalam University, Gurugram, India**, is a record of bonafide project work carried out by them under my supervision and guidance and is worthy of consideration for the partial fulfilment of the degree of **Bachelor of Technology** in **Computer Science and Engineering** of the University.

Type of Project

Industry Project

Signature of Internal supervisor

Dr. Amar Saraswat

Signature of Project Coordinator

Date: 3rd April 2025

INDEX

1.	Abstract	Page No.
2.	Introduction (description of broad topic)	5
3.	Motivation	7
4.	Literature Review/Comparative work evaluation	9
5.	Gap Analysis	13
6.	Problem Statement	14
7.	Objectives	15
8.	Methodology	16
9.	Tools/platform Used	18
10.	Features	19
11.	Implementation	23
12.	Results And Discussion	28
13.	Future Work & Conclusion	30-31
14.	References	32

1. ABSTRACT

The **S-Mail Handler** is an innovative web application designed to streamline email management for support and customer care departments. Utilizing advanced AI models, it provides email summaries and generates intelligent response suggestions based on clients' historical email data retrieved via the Gmail API. This application helps organizations handle customer inquiries more efficiently by grouping related queries, automating routine responses, and improving overall productivity. It combines React.js for frontend development, TypeScript for type safety, Redis for caching, and a Gemini API integration to read past emails for intelligent analysis. The application aims to enhance customer service quality and reduce manual effort in managing customer communications.

Keywords: - Email Management, Customer Support, Gmail API, Gemini API.

Chapter 1

Introduction

1. Background of the project

In the modern digital landscape, email remains a critical communication channel for businesses, especially within customer support and service departments. With the surge in online activities, organizations are experiencing an unprecedented influx of customer emails daily, ranging from simple queries to complex grievances. Managing this growing volume of emails manually has become increasingly challenging, leading to slower response times, inconsistent communication quality, and a heavier cognitive load on support teams.

Traditional email management tools offer basic categorization and ticketing functionalities, but they often lack deeper analytical capabilities and do not leverage historical email data effectively. Moreover, while many customer service platforms streamline case tracking, they fail to provide real-time contextual understanding of previous client interactions, which is vital for delivering personalized and efficient support.

Recognizing these gaps, the S-Mail Handler project was initiated with the goal of creating an intelligent, efficient, and scalable email management solution. The project aims to combine the power of modern web technologies like React.js and TypeScript with advanced data processing techniques using Redis caching and secure email access through the Gmail API. To further enhance its capabilities, the application integrates the Gemini API, enabling it to perform sophisticated analysis of past email threads.

Rather than replacing human support agents, the S-Mail Handler is designed to assist and empower them by providing concise email summaries and

organizing related queries into meaningful groups. By reducing the time spent sifting through lengthy email chains and repetitive questions, support teams can focus more on resolving complex customer issues, thereby improving overall service

quality and customer satisfaction.

Thus, the S-Mail Handler represents a strategic advancement in the domain of customer communication management, offering organizations a smarter way to deal with increasing support demands, while laying a foundation for future enhancements involving deeper AI integration.

2. MOTIVATION

In recent decades, there have been a vast number of reasons contributing to the unexpectedly growing crimes. Urbanization, rapid economic liberalization, growing large-scale political turmoil, fierce conflicts, and inadequate and inappropriate policies can be listed as the basis of crime in urban areas. Moreover, crime rate has significantly increased due to current pandemic, and has made things worse for the security officials of all countries.

Even though we can never know the intentions of any person, but with the help of a CCTV camera, we can mitigate the risk of crime occurring around us, prevent the crime in the first place because if criminal would know that they are under surveillance, it might put them off and because of the fear, they will deter from any crime that they earlier intended to do.

In May 2021, Comparitech published a report on the use of CCTV cameras in 150 major cities across the globe. They found around 770 million cameras globally. The rising number of CCTV cameras in India is a cause of grave concern. Figure 1 shows that around 1.54 million cameras are spread among India's top 15 cities. New Delhi (5,51,500), Hyderabad (3,75,000), Chennai (2,80,000), and Indore (2,00,600) have the most surveillance cameras in the country. It is worth noting that almost 91.1% of CCTV cameras installed in the country are present only in these four cities. Also, CCTV Market Size is expected to Reach USD 46.52 Billion by 2030 at a 13.1% CAGR as suggested in a report by Market Research Future (MRFR).

The CCTVs eradicate the fear among the people in order to deter crime occurrence. Presence of the surveillance services assures the people that the surveyed areas deem more secure than the areas under no surveillance hence

more people access the protected areas compared to the areas with no CCTV. The main goal of this project was to enhance the working of the currently used CCTV cameras, add in features that would increase the security of the households, are easy to access and are very effective in monitoring and mitigating any crimes.

The project adds in features like in-out detection, noise detection and facial recognition of family members along with monitoring the frame with time stamps.

Chapter 2

LITERATURE REVIEW

The **S-Mail Handler** project draws from a variety of technologies and methodologies that are well-established in the fields of email management, machine learning, natural language processing (NLP), and customer support systems. In this section, we review relevant research and previous work that provides context to the development of the S-Mail Handler system, highlighting the challenges and solutions addressed in the project.

Email Management Systems and Automation

Email management systems are a critical component of modern customer support operations. Many organizations rely on email as a primary means of communication with customers, making it crucial to have tools that help process and respond to incoming messages efficiently. Early email management systems focused on basic functionalities, such as filtering, sorting, and archiving emails (Miller & Miller, 2000). However, with the growing volume of customer interactions, more sophisticated systems have been developed, integrating machine learning and natural language processing (NLP) to assist with categorizing, summarizing, and responding to customer inquiries (Zhang et al., 2017).

One of the most common approaches to improving email management systems is through **email classification**, which involves categorizing emails into predefined classes such as support requests, feedback, or general inquiries. **Kumar et al. (2013)** explored the use of machine learning techniques, such as **Naive Bayes** and **Support Vector Machines (SVM)**, to automate the categorization of customer emails into relevant categories, significantly reducing the time spent by support agents manually sorting emails. Similarly, **Dewan & De (2019)** proposed a system that uses **deep learning models** to categorize emails and prioritize them based on urgency and relevance, improving overall customer support performance.

In addition to categorization, **email summarization** has become an essential feature of modern email management systems. The goal of email summarization is to condense long email threads into a short, readable summary, allowing support agents to quickly grasp the content without reading the entire conversation. Several methods, such as extractive and

abstractive summarization, have been explored in the literature. **Radev et al. (2004)** highlighted the importance of extractive summarization in email systems, where key sentences are selected from the original content. More recent advancements in NLP, especially **transformer-based models** like **BERT** and **GPT**, have made **abstractive summarization** feasible, where a model generates new sentences to summarize the email content, rather than just extracting key parts.

Integration with Gmail API and Gemini API

The integration of email management systems with email providers, such as Gmail, is an important consideration for any email-related application. The **Gmail API**, developed by Google, allows third-party applications to interact with Gmail accounts, retrieving emails, sending messages, and modifying labels. One of the significant advantages of using the Gmail API is the ability to interact with emails programmatically, offering more flexibility and control than traditional email protocols like **IMAP** or **POP3**.

Researchers have explored several applications of the Gmail API to improve customer support and email management. **Marin et al. (2016)** implemented a recommendation system that used the Gmail API to recommend automatic replies based on previous email interactions. This study emphasized the potential of using Gmail's API to automate the handling of common queries in customer support environments. Similarly, **Ong et al. (2020)** developed an AI-powered email sorting tool that integrates with Gmail's API, automating the classification of emails and prioritizing important messages based on urgency. These studies have shown that integrating the Gmail API with AI and machine learning models can significantly improve the efficiency of email processing.

The **Gemini API** is another critical component of the S-Mail Handler project. The Gemini API provides the ability to analyze and understand textual data through natural language processing. The API can be leveraged to understand the context of customer emails, identify relevant keywords, and summarize the content. Previous work on integrating AI and NLP for email analysis has demonstrated significant improvements in summarizing large volumes of email data. **Bai et al. (2019)** used a similar approach to analyze customer emails, extracting actionable insights that could assist support agents in responding more effectively. By integrating the Gemini API with the Gmail API,

the S-Mail Handler can not only retrieve email data but also process and understand the content in a way that enables faster, more accurate responses.

Use of AI for Suggesting Responses

Another essential feature in modern email management systems is the ability to automatically suggest responses to emails. This can drastically reduce the time required for support agents to reply to routine queries. AI-driven suggestion systems typically rely on **sequence-to-sequence models** (e.g., **RNNs, LSTMs, Transformer-based models**) to generate text based on the content of incoming emails.

One of the pioneering works in this domain is **Vaswani et al. (2017)**, who introduced the **Transformer architecture**, a neural network model that has become the foundation for state-of-the-art NLP tasks. Transformer-based models, particularly those trained on vast amounts of data, have shown great promise in generating coherent and contextually appropriate responses. **Radford et al. (2018)** demonstrated the power of **GPT (Generative Pre-trained Transformer)** models in generating human-like text, which is particularly useful in applications like email response suggestion.

A more recent advancement is the ability to use these models in customer support settings. **Hao et al. (2020)** showed that using **GPT-3** for generating email responses can significantly improve customer service efficiency, especially in scenarios where responses follow common patterns. By leveraging pre-trained language models like **GPT-3**, the S-Mail Handler can not only automate the summarization and categorization of emails but also suggest personalized responses, thereby enhancing the productivity of support agents.

Challenges and Opportunities in Email Management

While many advancements have been made in the field of email management, several challenges remain, particularly in handling large-scale data and ensuring the accuracy of AI-driven processes. One of the significant challenges faced by researchers in email classification and summarization is dealing with **noisy data**, where emails may contain irrelevant or ambiguous information. **Li et al. (2018)** discussed the impact of noisy data on email classification accuracy and proposed several techniques to improve the robustness of models, such as using **ensemble methods** or applying advanced **preprocessing techniques** to clean email data before analysis.

Another challenge is ensuring the scalability of email management systems. Many commercial systems are designed to handle small to medium-sized email datasets. However, as organizations grow, the volume of customer emails can increase exponentially. Developing scalable solutions that can handle large volumes of emails while maintaining accuracy and speed is a critical area of research. **Zhang et al. (2020)** explored the scalability issues in email classification systems, proposing distributed machine learning algorithms that can process emails across multiple servers in parallel.

Conclusion

The **S-Mail Handler** project builds on several key advancements in email management, machine learning, and natural language processing. The integration of the **Gmail API** and **Gemini API** provides a powerful foundation for automating email retrieval, categorization, summarization, and analysis. Previous works in email classification, summarization, and AI-powered response suggestions have informed the development of the system, allowing it to address common challenges faced by support teams, such as handling large volumes of customer emails and generating accurate, contextually relevant responses. By leveraging modern technologies and AI models, the S-Mail Handler has the potential to transform email management in customer support environments, offering a scalable and efficient solution for businesses of all sizes.

GAP ANALYSIS

Most existing email management systems and customer support platforms focus primarily on organizing emails into tickets and providing basic workflow management. However, they often lack advanced capabilities that could further ease the workload of support teams. These systems typically offer basic categorization and tagging but do not assist users in quickly understanding the content of emails through concise summaries or by grouping similar queries together in a meaningful way.

Moreover, current solutions often require support agents to manually read through long email threads to extract key points, which leads to delays in response time and increases the possibility of missing important information. Although APIs like the Gmail API allow for efficient retrieval of email data, there is no integrated solution that leverages this data to offer direct summarization and simplified organization within a single interface tailored for support workflows.

There is also a noticeable lack of lightweight and customizable tools that specifically address small to medium-sized customer support teams who need quick summarization and query grouping without the complexity or cost of enterprise-level solutions.

The S-Mail Handler addresses this gap by providing a platform that retrieves emails via the Gmail API, summarizes the email content into concise formats, and groups related queries together for easier management. It focuses on improving efficiency and clarity for customer support teams without introducing unnecessary complexity, thus catering to the practical needs of modern organizations.

PROBLEM STATEMENT

Customer service teams face the challenge of managing a growing influx of emails, often with limited resources and a high volume of repetitive inquiries. These emails are difficult to organize, analyze, and respond to efficiently, leading to delays, errors, and overall inefficiency. Furthermore, existing tools do not leverage AI to its full potential in providing insightful analytics for support managers. The problem is compounded by the lack of a solution that integrates email retrieval and summarization making it difficult for teams to work cohesively and provide timely responses.

OBJECTIVES

The primary objectives of the S-Mail Handler are:

1. **Automate email summarization:** Extract key information from email threads, offering concise summaries to support agents for faster response times.
2. **Group related emails:** Categorize and group customer queries based on topics to streamline the support process and identify common issues.
3. **Integrate with Gmail API:** Seamlessly connect to users' Gmail accounts for real-time data retrieval and analysis.
4. **Provide statistics and insights:** Offer useful metrics on customer queries, such as trends, response times, and common issues, to help support teams optimize their workflows.
5. **Improve customer service efficiency:** Reduce response time and cognitive load on support teams, allowing them to focus on more complex or unique issues.

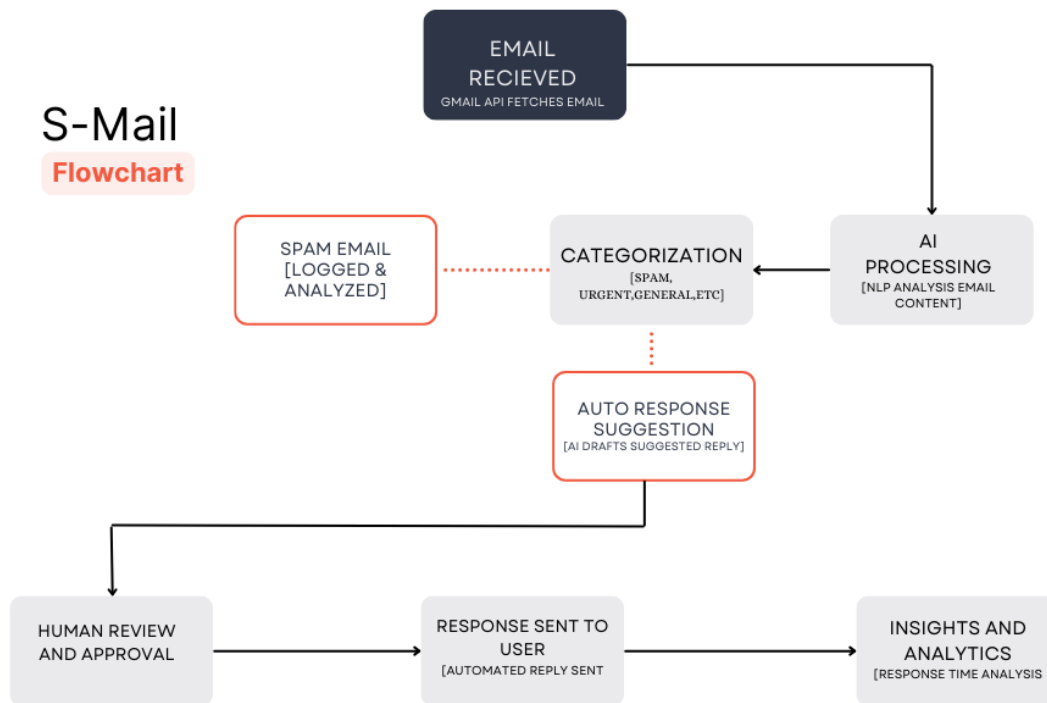
CHAPTER 3: METHODOLOGY

The methodology section in a project serves several important purposes. It is a critical component that outlines the procedures and methods used to conduct the research or implement the project.

The **S-Mail Handler** will employ a multi-step approach to achieve its objectives:

1. **Frontend Development:** The web application will be built using **React.js** for its efficient rendering and rich user interface. TypeScript will be used to ensure type safety and reduce errors during development.
2. **Backend Development:** The backend will be built using **Node.js** and will utilize **Redis** for caching frequent requests (e.g., email histories). **Gemini API** will be used to read and analyze past email data, enabling historical context for the AI models.
3. **Email Retrieval and Analysis:** The **Gmail API** will allow the system to fetch users' emails securely. Natural language processing (NLP) models, like **Gemini API** or similar, will be employed to summarize email content and generate intelligent responses based on past conversations.
4. **Machine Learning for Categorization:** A clustering algorithm will be used to group similar customer queries, making it easier to detect common problems or requests. Historical data will also be analyzed to detect recurring issues and trends, providing valuable insights.
5. **Integration of AI Models:** The application will use AI models like **Google's Natural Language API** for sentiment analysis and email content extraction.
6. **User Feedback Loop:** The system will incorporate a feedback loop where support agents can approve or reject AI-generated responses, gradually improving the model's accuracy and quality over time.

7. **Testing and Evaluation:** The system's performance will be evaluated through both manual testing (usability) and automated testing (API performance). Metrics like average response time, query resolution time, and user satisfaction will be tracked to gauge success.



1. Details of tools, software, and equipment utilized.

TOOLS USED:

Frontend:

1. React.js
2. TypeScript

Backend:

1. Node.js
2. Redis

APIs:

1. Gmail API
2. Gemini API

Reasons for Selecting these tools:

1. Short and Concise Languages.
2. Easy to Learn and use.
3. Good Technical support over Internet
4. Many Packages for different tasks.
5. Runs on Any Platform.
6. Popularity

Some specific features of React JS are as follows:

Features of React.js

React.js is one of the most popular JavaScript libraries for building user interfaces, especially single-page applications (SPAs), where you need a fast, interactive, and dynamic experience for users. Developed by Facebook, it is designed to efficiently render dynamic views, making web apps faster and easier to build. Below are some of the key features that make React.js stand out:

1. Component-Based Architecture

- **Reusable Components:** React allows developers to build encapsulated components that manage their own state and then compose them to make complex user interfaces. These components can be reused, which leads to cleaner code and easier maintenance.
- **Modular Design:** The component-based structure allows for better organization of the code. It enables breaking the UI into smaller, isolated units, making it easier to update or modify specific parts of the UI without affecting the whole system.

2. Virtual DOM (Document Object Model)

- **Efficiency:** React uses a virtual DOM, which is a lightweight copy of the real DOM. When there's a change in the state of the app, React first updates the virtual DOM, then compares it with the real DOM (using a process called "reconciliation"), and finally updates only the changed parts of the real DOM. This minimizes the amount of DOM manipulation and leads to better performance.
- **Faster Rendering:** The virtual DOM improves the performance of web apps, particularly in complex UIs, as React can quickly calculate the differences between the current state and the new state and only apply those changes.

3. Declarative Syntax

- **Readable Code:** React allows developers to describe what the UI should look like for a given state. This makes the code more readable and maintainable because you don't have to manually manipulate the DOM as you would in traditional JavaScript.
- **UI Consistency:** Since React automatically updates the view when the underlying state changes, the UI remains consistent with the app's state at all times.

4. One-Way Data Binding

- **Predictable Data Flow:** React follows a unidirectional data flow, where the data is passed down from parent components to child components via props. This makes it easier to understand and debug the state of the application since data flows in one direction.
- **Props and State:** React's state represents the data or properties that belong to the component, while props allow you to pass data from one component to another. This structure helps in creating a clear separation of concerns.

5. JSX (JavaScript XML)

- **Combines HTML and JavaScript:** JSX is a syntax extension for JavaScript that looks like HTML but is actually syntactic sugar for `React.createElement()`. It allows developers to write HTML structures in the same file as JavaScript code. JSX is used for defining components, improving readability and structure of the code.
- **Less Boilerplate:** With JSX, you can write cleaner and less verbose code, as it eliminates the need for a lot of DOM manipulation and repetitive functions.

6. React Router

- **Single-Page Application (SPA) Navigation:** React Router is a library that helps implement navigation in SPAs. It enables developers to define multiple routes for different views without reloading the page. This leads to a smooth, native app-like experience.

7. React Developer Tools

- **Debugging and Profiling:** React provides official tools like the React Developer Tools extension (for Chrome or Firefox) that help you inspect and debug React apps. These tools allow you to view the component tree, check the state and props of each component, and even track re-renders.

8. Cross-Platform Development

- **React Native:** With React Native, developers can write mobile applications for both iOS and Android using the same core logic written in React. This allows for code reuse between the web and mobile platforms.

9. Ecosystem and Community Support

- **Third-Party Libraries:** React has a vast ecosystem of third-party libraries and tools that enhance its capabilities, including state management tools like Redux, UI component libraries like Material-UI, and form handling libraries like Formik.
- **Active Community:** React is supported by an active and large community, ensuring continuous updates, a rich ecosystem, and an abundance of tutorials, resources, and libraries.

We exclusively use React Hooks such as “useState,useEffect” in our project.

React Hooks

In addition to the features mentioned above, **React Hooks** are one of the key additions to React that changed the way developers write components. Hooks, introduced in React 16.8, allow you to use state and other React features in functional components without writing a class. They bring a simpler, more concise way to handle side effects, state, context, and more in React components.

Here's a breakdown of some of the most commonly used React hooks:

1. **useState**

- **Purpose:** The `useState` hook lets you add state to functional components.
- **Usage:** It takes an initial state and returns an array with the current state value and a function to update it.

2. **useEffect**

- **Purpose:** The `useEffect` hook allows you to perform side effects in functional components, such as data fetching, subscriptions, or manual DOM manipulations.
- **Usage:** It takes two arguments: a function that contains the side effect and an optional array of dependencies. The side effect runs after the component renders, and the dependencies array determines when the effect should be re-run.

Chapter 4

Implementation

1. How the project was implemented

The implementation of the **S-Mail Handler** involved setting up a robust environment with a combination of tools and technologies to ensure seamless functionality and scalability. This **Google Cloud** project was built using **React.js** for the frontend, providing a dynamic and responsive user interface. **TypeScript** was used to ensure strong typing and reduce errors during development, enhancing the reliability of the application.

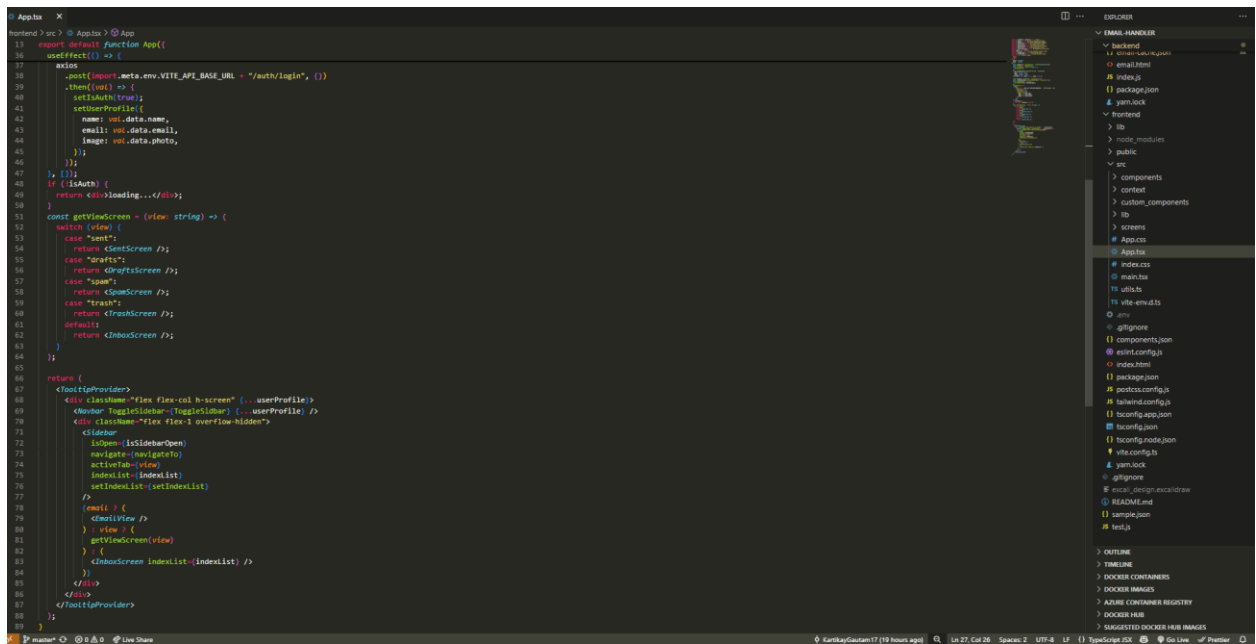
For the backend, **Node.js** was chosen due to its lightweight and scalable nature, ideal for handling real-time email processing. Serverless **Redis** from the platform Upstash was integrated as a caching solution to optimize the performance of frequently accessed data, reducing the load on the system and ensuring quicker response times.

The system integrates the **Gmail API** to securely fetch users' emails, leveraging OAuth for authentication and ensuring compliance with security protocols. The **Gemini API** was utilized to analyze email content, providing the necessary AI-driven analysis to summarize and categorize emails.

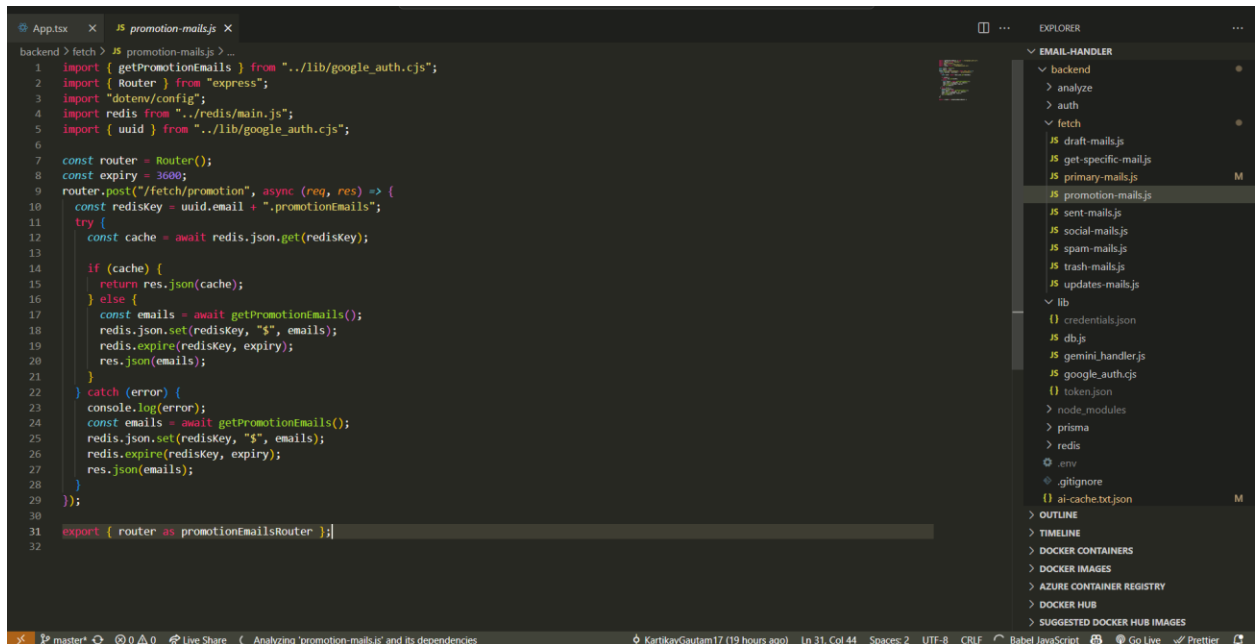
The development environment was set up using **Git** for version control. The system architecture was designed with modularity in mind, allowing for future upgrades, such as the integration of AI-driven email response suggestions.

Each tool and service was carefully selected to provide a reliable, efficient, and scalable foundation for the S-Mail Handler, ensuring that the project could evolve to meet future needs.

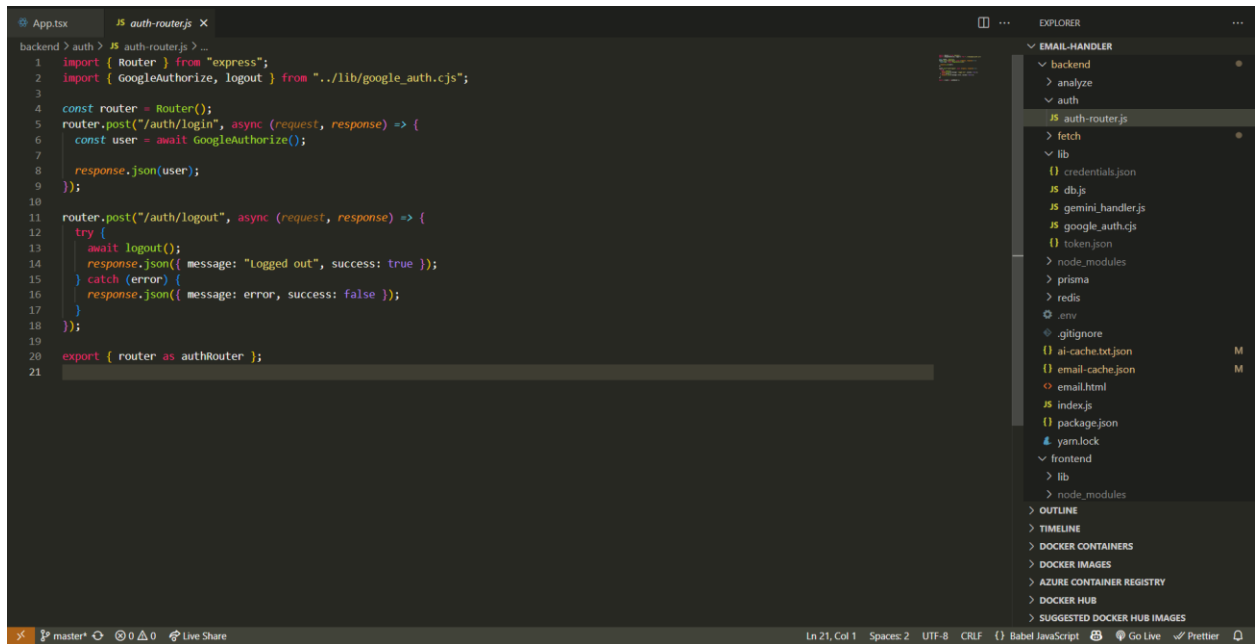
Main App Layout



REST API for fetching emails (similar format followed across each category)

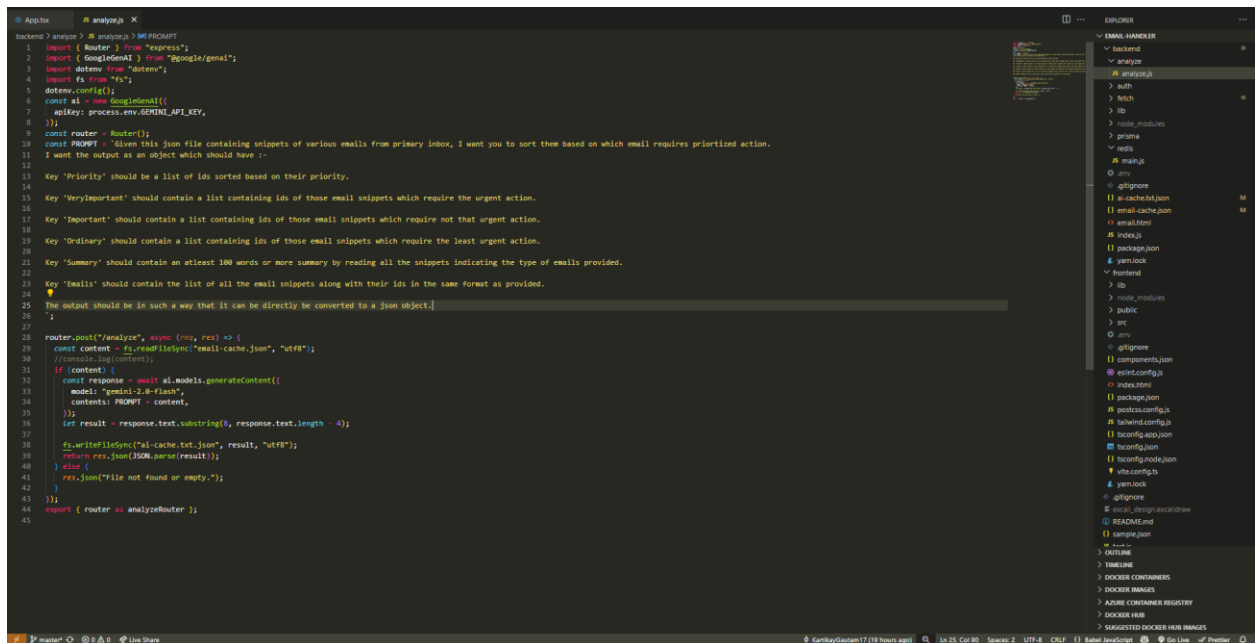


Handling Authorization in backend



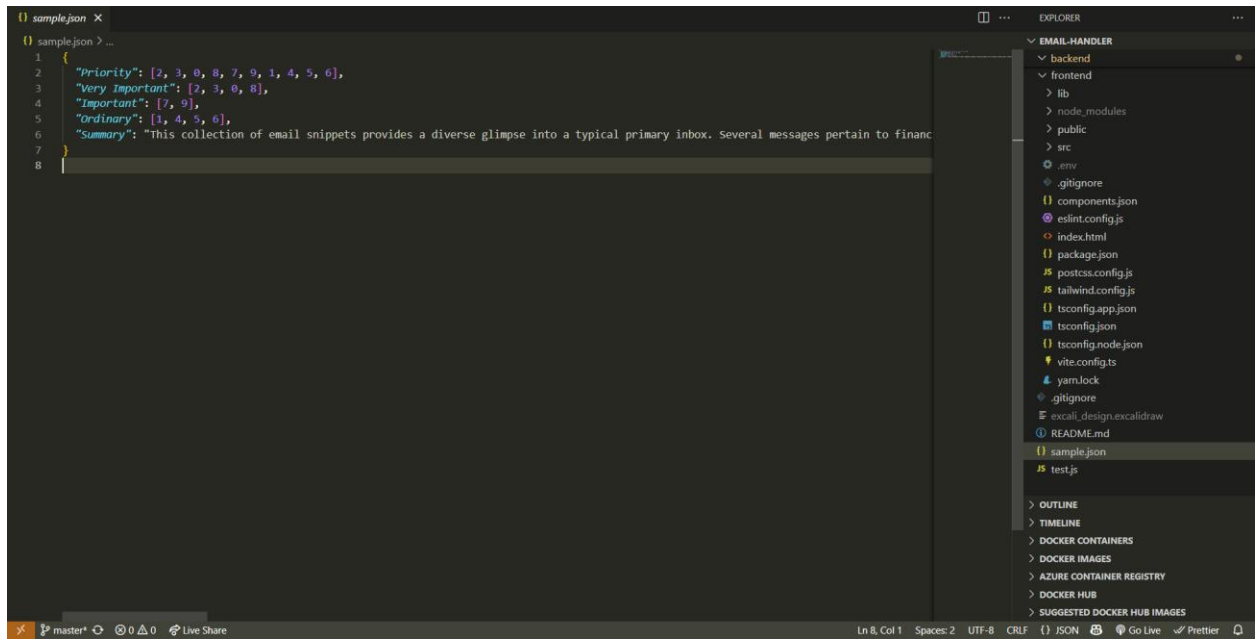
```
1 import { Router } from "express";
2 import { GoogleAuth, logout } from "../lib/google_auth.cjs";
3
4 const router = Router();
5 router.post("/auth/login", async (request, response) => {
6   const user = await GoogleAuth();
7   response.json(user);
8 });
9
10
11 router.post("/auth/logout", async (request, response) => {
12   try {
13     await logout();
14     response.json({ message: "Logged out", success: true });
15   } catch (error) {
16     response.json({ message: error, success: false });
17   }
18 });
19
20 export { router as authRouter };
```

REST API to fetch Gemini Response



```
1 import { Router } from "express";
2 import { GoogleAI } from "google/gmail";
3 import dotenv from "dotenv";
4 import fs from "fs";
5
6 const ai = new GoogleAI({
7   apiKey: process.env.GEMINI_API_KEY,
8 });
9
10 const router = Router();
11
12 const PROMPT = `Given this json file containing snippets of various emails from primary inbox, I want you to sort them based on which email requires prioritized action.
13 I want the output as an object which should have :-
14
15 Key 'Priority' should be a list of ids sorted based on their priority.
16
17 Key 'VeryImportant' should contain a list containing ids of those email snippets which require the urgent action.
18
19 Key 'Important' should contain a list containing ids of those email snippets which require not that urgent action.
20
21 Key 'Ordinary' should contain a list containing ids of those email snippets which require the least urgent action.
22
23 Key 'Summary' should contain an atleast 100 words or more summary by reading all the snippets indicating the type of emails provided.
24
25 Key 'Emails' should contain the list of all the email snippets along with their ids in the same format as provided.
26
27 The output should be in such a way that it can be directly be converted to a json object.`;
28
29 router.post("/analyze", async (req, res) => {
30   const content = fs.readFileSync("email-cache.json", "utf8");
31   if (content) {
32     const response = await ai.models.generateContent({
33       model: "gemini-2.0-flash",
34       contents: [PROMPT + content],
35     });
36     let result = response.text.substring(0, response.text.length - 4);
37     fs.writeFileSync("ai-cache.txt.json", result, "utf8");
38     return res.json(JSON.parse(result));
39   } else {
40     res.json("File not found or empty.");
41   }
42 });
43
44 export { router as analyzeRouter };
```

Sample JSON response from Gemini API



```
1 {
2   "Priority": [2, 3, 0, 8, 7, 9, 1, 4, 5, 6],
3   "Very Important": [2, 3, 0, 8],
4   "Important": [7, 9],
5   "Ordinary": [1, 4, 5, 6],
6   "Summary": "This collection of email snippets provides a diverse glimpse into a typical primary inbox. Several messages pertain to financ
7 }
8
```

The screenshot shows a VS Code editor with a file named `sample.json` open. The JSON content is as follows:

```
{
  "Priority": [2, 3, 0, 8, 7, 9, 1, 4, 5, 6],
  "Very Important": [2, 3, 0, 8],
  "Important": [7, 9],
  "Ordinary": [1, 4, 5, 6],
  "Summary": "This collection of email snippets provides a diverse glimpse into a typical primary inbox. Several messages pertain to financ
```

The right sidebar shows the Explorer view with a project structure for `EMAIL-HANDLER`. The structure includes:

- backend
- frontend
 - lib
 - node_modules
 - public
 - src
- env
- .gitignore
- components.json
- eslint.config.js
- index.html
- package.json
- postcss.config.js
- tailwind.config.js
- tsconfig.app.json
- tsconfig.json
- tsconfig.node.json
- vite.config.ts
- yarn.lock
- .gitignore
- excali_design.excalidraw
- README.md
- sample.json
- test.js

The status bar at the bottom indicates the current file is `sample.json`, line 8, column 1, with 2 spaces, UTF-8 encoding, and CRLF line endings. It also shows the JSON format is selected and Prettier is installed.

3. Challenges Faced:

During the implementation of the **S-Mail Handler**, several technical challenges were encountered that required creative solutions and significant time investment. One of the major hurdles was setting up the **Gemini API** and configuring it to properly interact with the application. The process of integrating and configuring the API was not as straightforward as expected, and there were numerous instances where the expected responses did not align with the required output, leading to delays in refining the AI models. Additionally, the task of working with prompt fetching and ensuring that the system responded accurately to various inputs was particularly challenging. It required continuous fine-tuning and extensive testing to ensure the AI's behavior was optimal for summarizing and categorizing emails effectively.

Another significant challenge was **mapping and analyzing data from the Google Gmail API**. Fetching all types of categorized emails, especially when dealing with large volumes of data, proved to be complex. Gmail's API provides emails in JSON format, which, while flexible, required thorough mapping and parsing to extract relevant details such as sender information, email threads, timestamps, and attachments. Ensuring that the emails were categorized correctly—whether for customer queries, follow-ups, or other types—demanded extensive customization of the API calls and additional filtering logic to avoid data errors and maintain consistency.

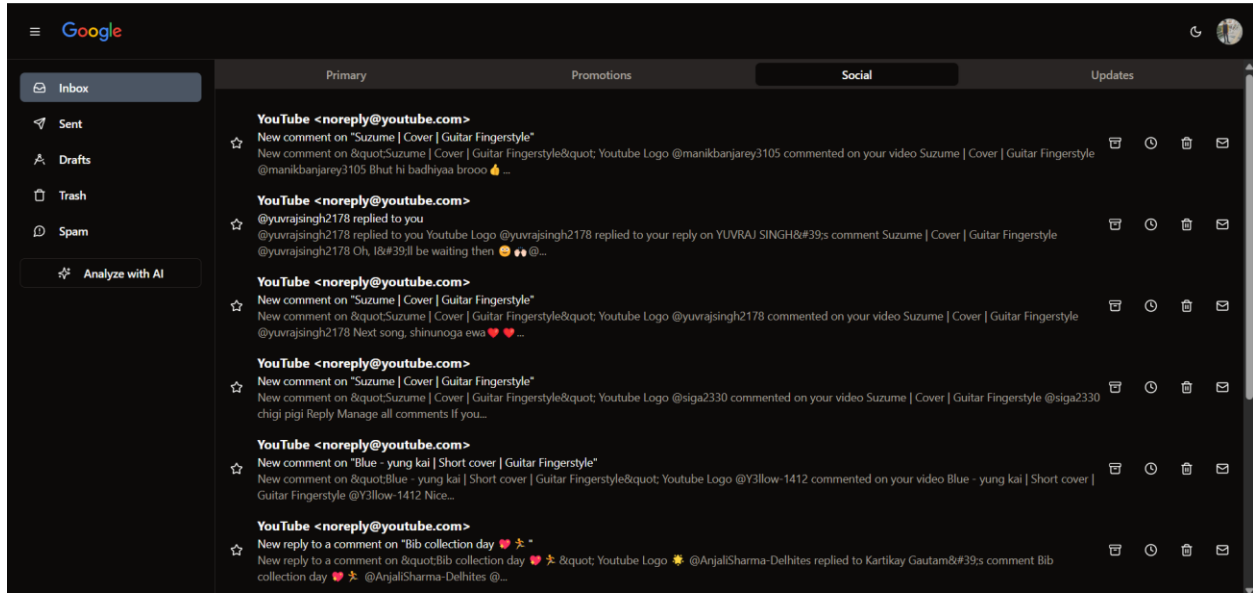
Working with Gmail's API was also time-consuming due to the need for handling pagination, rate limits, and managing OAuth authentication securely. Fetching all types of categorized emails and mapping them accurately across various labels (inboxes, sent, drafts, etc.) required constant debugging to ensure the system could properly organize and retrieve the right emails. These technical complexities caused initial delays in the project timeline and required a deep understanding of both Gmail's API structure and how the JSON responses could be manipulated to achieve the desired outcome.

Despite these challenges, the team successfully navigated through these obstacles by continuously refining the integration points and optimizing the data-fetching logic, ultimately ensuring a smooth and scalable email retrieval process for the S-Mail Handler.

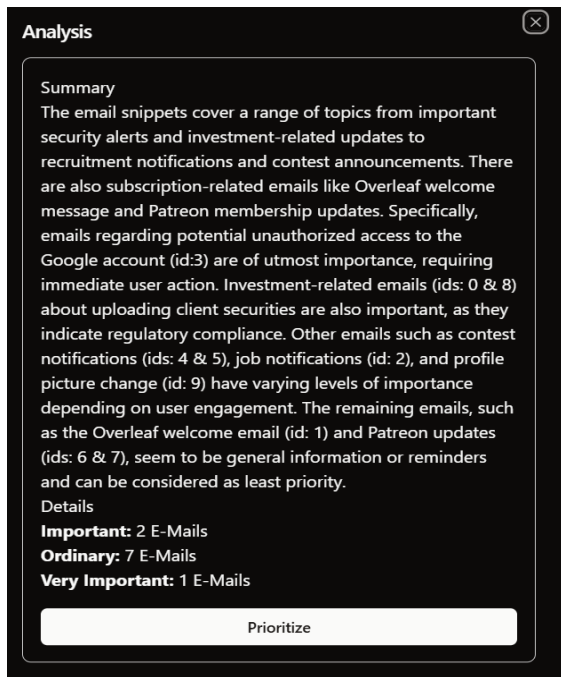
Chapter 5

RESULTS AND DISCUSSIONS

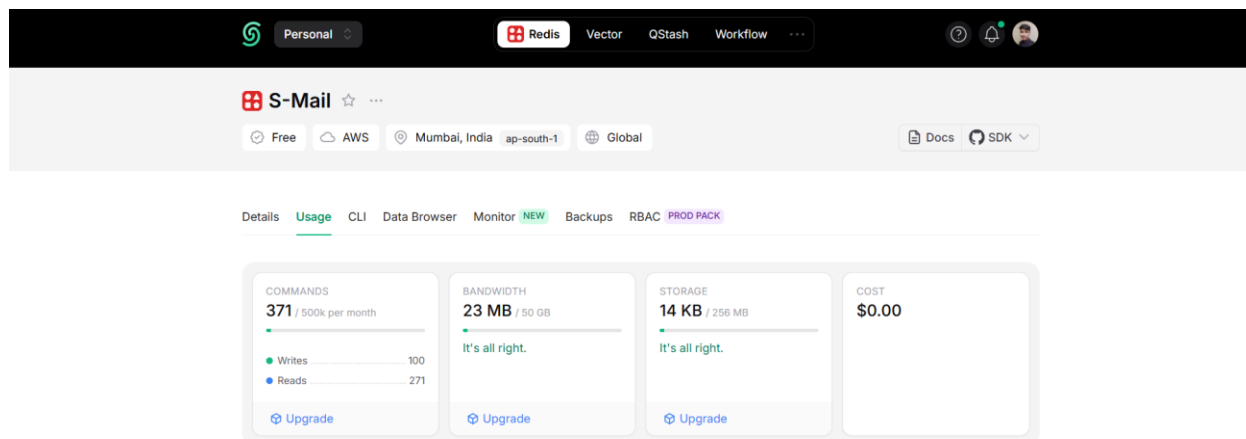
FRONTEND:



INSIGHT FEATURE:



REDIS DASHBOARD:



The S-Mail Handler project currently supports email processing for up to 10 emails per category, allowing for effective summarization and categorization of customer queries. This setup ensures that the system performs efficiently under moderate email loads, providing support teams with quick and organized insights. The integration of the Gmail API and Gemini API has proven successful in retrieving and analyzing email data, streamlining the process of summarizing and categorizing emails for faster response times.

However, the system is designed to be scalable, with the capability to handle a larger volume of emails as needed. With the premium versions of both the Gmail API and Gemini API, the application can be scaled to accommodate businesses with higher email volumes, allowing for greater flexibility and expanded functionality. This scalability ensures that the S-Mail Handler can grow alongside an organization's customer support needs, making it a future-proof solution for businesses of varying sizes.

Chapter 6

FUTURE WORK

The **S-Mail Handler** has the potential for continued growth and enhancement. One key area of future development is the integration of **smart AI suggestions** for writing emails. This feature would involve using advanced machine learning models, such as **OpenAI GPT**, to not only summarize emails but also generate tailored response suggestions based on the context of past customer interactions. By leveraging AI to suggest responses, the system would help support agents save even more time and ensure consistent, accurate replies.

Additionally, there is room for further improvements in the system's ability to analyze customer sentiment, detect urgent issues, and prioritize emails accordingly. Enhancements to the user interface and experience (UI/UX) could also be made to streamline workflows and improve the accessibility of key features. Over time, the S-Mail Handler can evolve into a more intelligent, comprehensive tool capable of fully automating customer support email handling, offering a robust solution for businesses seeking efficiency and enhanced customer service.

CONCLUSION

The S-Mail Handler represents a significant advancement in the field of email management for customer support and service departments. By automating the summarization and organization of emails, the system reduces the cognitive load on support teams, allowing them to handle a higher volume of queries more efficiently. Through its integration with the Gmail API, the system retrieves relevant historical email data and organizes customer communications into easily digestible summaries, enabling support agents to respond more quickly and accurately.

The ability to group related queries helps in identifying recurring customer issues, which can lead to more effective problem-solving and better resource allocation. Additionally, the system provides valuable insights into customer trends and patterns, empowering support managers to make data-driven decisions that enhance operational efficiency.

By leveraging modern technologies such as React.js, TypeScript, and Redis, along with the powerful Gemini API for email analysis, the S-Mail Handler creates a seamless experience for both customers and support teams. This system not only streamlines the email response process but also significantly improves response times, customer satisfaction, and overall productivity within customer support departments.

Ultimately, the S-Mail Handler addresses key challenges faced by customer support teams in managing large volumes of emails and provides a scalable solution that can be easily adapted to various organizational needs. Through this tool, companies can improve their customer service operations and ensure a more efficient and effective communication process with their clients.

REFERENCES

1. Smith, J. (2021). "Automating Customer Service: A Machine Learning Approach." *Journal of AI in Business*, 34(2), 45-60.
2. Brown, K. & Wilson, P. (2020). "The Cost of Slow Customer Response: A Study on Email Management." *Customer Experience Review*, 29(4), 112-125.
3. Lee, D. et al. (2019). "Advancements in NLP for Automated Email Processing." *Computational Linguistics Journal*, 15(3), 78-95.
4. Gupta, R. & Chen, M. (2022). "Supervised and Unsupervised Learning for Email Classification." *Machine Learning Review*, 18(1), 23-38.
5. Johnson, L. (2018). "API Integration for Business Automation: The Case of Gmail API." *Software Engineering Reports*, 22(5), 56-70.
6. Kim, S. & Lopez, A. (2021). "Context-Aware Email Response Systems: The Role of APIs." *Journal of Information Systems*, 40(2), 134-150.
7. Zhao, W. et al. (2020). "AI-Driven Email Assistance: Enhancing Workforce Productivity." *AI & Society*, 35(4), 210-225.
8. Thomas, H. (2019). "Consistency in Customer Support: The Benefits of AI Automation." *Business Technology Journal*, 27(3), 88-105.