# GMAIL CLONE

**A PROJECT REPORT**
**for**
**Mini Project (KCA353)**
**Session (2023-24)**

**Submitted by**

**Ayush Narayan Maurya**
**(2200290140046)**

**Submitted in partial fulfilment of the**
**Requirements for the Degree of**

**MASTER OF COMPUTER APPLICATION**

**Under the Supervision of**
**Dr. Vipin Kumar**
**Associate Professor**

**Submitted to**

**DEPARTMENT OF COMPUTER APPLICATIONS**
**KIET Group of Institutions, Ghaziabad**
**Uttar Pradesh-201206**
**(JANUARY 2024)**

# DECLARATION

I hereby declare that the work presented in this report entitled "**Gmail Clone**", was carried out by me. I have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute. I have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results, that are not my original contribution. I have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.

I affirm that no portion of my work is plagiarized, and the experiments and results reported in the report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, I shall be fully responsible and answerable.

**Name**: Ayush Narayan Maurya (2200290140046)

**(Candidate Signature)**

# CERTIFICATE

Certified that **Ayush Narayan Maurya** (2200290140046**)** have carried out the project work having "**Gmail Clone**" for Master of Computer Applications from Dr. A.P.J. Abdul Kalam Technical University (AKTU**)**, Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself / herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

**Date:**

**Ayush Narayan Maurya (**2200290140046)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

**Date:**

| | |
|---|---|
| **Dr. Vipin Kumar** | **Dr. Arun Tripati** |
| **Associate Professor** | **Head** |
| **Department of Computer Applications** | **Department of Computer Applications** |
| **KIET Group of Institutions, Ghaziabad** | **KIET Group of Institutions, Ghaziabad** |

# ABSTRACT

The Gmail Clone project aims to replicate the core functionalities of the popular email service Gmail while incorporating modern web technologies and user-friendly features. Email communication remains a vital part of personal and professional interactions, and a Gmail-inspired clone will address the need for a secure, efficient, and user-friendly email solution.

This project will leverage web development technologies to create a web-based email application that provides users with a seamless and intuitive email experience. The Gmail Clone project is a comprehensive endeavor to develop a full-fledged email application that replicates the core features and functionalities of Google's Gmail. Users can create accounts securely and log in with their credentials. Account authentication will follow best practices to ensure data security.

The application will allow users to compose emails with support for rich text formatting, attachments, and the ability to send messages to multiple recipients. Users will have an inbox to receive and manage their emails. They can organize messages with labels, folders, and utilize a robust search function to find specific emails. Emails will be grouped into threaded conversations, making it easier for users to follow and respond to ongoing discussions.

The Gmail Clone Web Application will be responsive, ensuring it functions seamlessly across various devices, including desktops, tablets, and smartphones. Robust search functionality and filtering options will be implemented to help users quickly locate specific emails and manage their inbox efficiently.

The Gmail Clone project aims to offer an open-source alternative to Gmail, delivering a secure, feature-rich, and customizable email experience. By focusing on user privacy, modern design, and efficient email management, this endeavor seeks to empower individuals and organizations to regain control over their email communication while ensuring a high level of user satisfaction.

# ACKNOWLEDGEMENT

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Dr Vipin Kumar** for his/ her guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Arun Kumar Tripathi**, Professor and Head, Department of Computer Applications, for his insightful comments and administrative help on various occasions.
Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

**Ayush Narayan Maurya**

# TABLE OF CONTENTS

# LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

## 1.1 OVERVIEW

The Gmail Clone project is a comprehensive initiative to replicate the core functionalities of Gmail, offering users a web-based email application with a familiar interface and feature set. The primary objective is to provide a seamless user experience while incorporating modern design principles, robust security measures, and scalability for future growth.

The project entails the development of a feature-rich email platform with functionalities such as composing, sending, receiving, and organizing emails with labels. The focus is on delivering a user-friendly application that mirrors the convenience and efficiency of Gmail, including support for attachments, inline images, and rich text formatting.

In terms of security, the project prioritizes user data protection through a robust user authentication system and the implementation of encryption for data in transit (HTTPS) and at rest (database). Security measures extend to addressing common web vulnerabilities through secure coding practices, ensuring a secure communication environment. Protective measures against Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF) further fortify the application against potential threats.

The user interface is designed to be responsive, adapting seamlessly to various screen sizes and devices. Leveraging modern frontend technologies such as React.js, the project aims to provide a clean, intuitive design that enhances the overall user experience.

On the backend, the project utilizes Node.js with Express.js to build a scalable RESTful API architecture. The use of MongoDB as the database offers flexibility and scalability for efficient data storage. The project also implements JSON Web Tokens (JWT) for secure user authentication, ensuring the protection of user accounts and data.

The development process follows a structured timeline, beginning with planning and design, where project requirements are defined, and wireframes are created. Frontend development involves the implementation of the user interface using React.js, while backend development focuses on server-side logic, APIs, and user authentication. Integration and testing phases ensure the seamless collaboration of frontend and backend components, with thorough testing to validate functionality. The deployment phase involves hosting the application and setting up monitoring and logging for ongoing maintenance.

## 1.2 OBJECTIVE

The Gmail Clone project aims to replicate Gmail's key features, emphasizing user-friendly design, security, and scalability. Objectives include implementing core email functionalities, ensuring secure user authentication, designing a responsive interface, and utilizing modern technologies.

The project targets scalability for future growth, a contemporary user interface with rich features, and robust security measures, addressing vulnerabilities like  Deployment, monitoring, comprehensive documentation, and rigorous testing further contribute to delivering a reliable, secure, and efficient web-based email application.

The Gmail Clone project is driven by a set of comprehensive objectives designed to create a robust and feature-rich web-based email application. These objectives span various facets of development, security, usability, and scalability to ensure the successful replication of Gmail's functionalities while incorporating modern design and technology standards.

Usability across devices is crucial, and the project aims to achieve this by implementing a responsive design. The user interface should seamlessly adapt to various screen sizes and devices, ensuring a consistent and user-friendly experience whether accessed from desktops, tablets, or mobile devices.

Utilizing modern and efficient technologies is crucial for the project's success. The chosen stack includes React.js for the frontend, Node.js with Express.js for the backend, and MongoDB for flexible data storage. This ensures a scalable, maintainable, and performant application that aligns with current industry standards.

## 1.3 PROJECT FEATURES

### 1.3.1 User Authentication and Authentication:

- Users can register for an account securely.
- Robust user authentication system ensures account security.

### 1.3.2 Compose and Send Emails:

- Users can compose emails with support for rich text formatting.
- Sending emails includes options for attaching files and inline images.

### 1.3.3 Receive and View Emails:

- Users can receive and view emails in an organized and user-friendly interface.
- Conversations are grouped to enhance readability.Quiz scheduling for future release.

### 1.3.4 Search Functionality:

- Robust search functionality allows users to quickly locate specific emails.
- Search filters enable refined searches based on criteria such as sender, subject, or date**.**

### 1.3.5 User Contacts:

- Users can manage a list of contacts, including adding, editing, and deleting contacts.
- Auto-suggestion for email addresses simplifies the process of adding recipients.

### 1.3.6 Responsive Design:

- The user interface is responsive, adapting to various devices and screen sizes.
- Ensures a consistent and user-friendly experience on desktops, tablets, and mobile devices.

### 1.3.7 Security Measures:

- Implementation of HTTPS for secure communication.

- Protection against common web vulnerabilities, including Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF).

### 1.3.8 Scalability:

- The architecture is designed to be scalable, accommodating an increasing user base.
- Optimization strategies for performance and resource management.

### 1.3.9 Attachments and Inline Images:

- Support for attaching files to emails.
- Inline image support for visually enriched email content.

### 1.3.10 User Interface Design:

- A clean and modern user interface design.
- Intuitive navigation and visually appealing layout for enhanced user experience.

### 1.3.11 Settings and Personalization:

- Users can customize account settings, including profile information and notification preferences.
- Personalization options for themes and display settings.

### 1.3.12 Logout and Session Management:

- Secure logout functionality to end user sessions.
- Session management to ensure security and prevent unauthorized access.

## 1.4 HARDWARE / SOFTWARE USED IN A PROJECT

### 1.4.1 Software Requirements

- OS - Windows 7,8,10,11
- Language – MERN Stack
- Platform - Web Browser

### 1.4.2 Hardware Requirements

- Processor – Dual Core and above
- RAM – 512 MB

- Storage – 20 GB
- Monitor – 15” Colour Monitor
- Keyboard – 122 keys

# CHAPTER 2

# LITERATURE REVIEW

A literature review is a critical analysis of existing literature on a specific topic or research question. In the context of a Gmail Clone project, the literature review may cover various aspects such as email platforms, web application development, user experience, security measures, and modern technologies. Here's a brief overview of potential areas to explore in a literature review for a Gmail Clone project:

## 2.1 Email Platforms and User Experience:

- Investigate existing literature on popular email platforms like Gmail, Outlook, and Yahoo Mail.
- Analyse studies and reviews on user experience design in email applications.
- Explore features that users find most valuable in email platforms.

## 2.2 Web Application Development Frameworks:

- Review literature on web development frameworks, with a focus on frontend frameworks like React.js.
- Explore best practices for creating responsive and user-friendly web interfaces.

## 2.3. Security in Web Applications:

- Investigate literature on web application security, including common vulnerabilities and best practices.
- Explore encryption methods for data in transit and at rest.
- Review studies on secure user authentication systems and protection against common web threats.

## 2.4. Scalability and Performance Optimization:

- Explore literature on designing scalable architectures for web applications.

- Investigate strategies for optimizing performance, handling increased user loads, and efficient resource management.
- Examine case studies of successful scalable web applications.

## 2.5 Technological Stack:

- Review literature on the technologies chosen for the project, such as React.js, Node.js, Express.js, and MongoDB.
- Explore case studies or research that highlight the advantages and challenges of using these technologies in web development.

## 2.6 User Authentication and Authorization:

- Investigate literature on secure user authentication mechanisms, including the use of JSON Web Tokens (JWT).
- Explore studies on best practices for user authorization and access control in web applications.

## 2.7 Responsive Design and User Interface:

- Review literature on responsive web design principles and techniques.
- Explore studies on creating modern and intuitive user interfaces.
- Investigate the impact of responsive design on user engagement and satisfaction.

## 2.8 Email Security and Encryption:

- Explore literature on email security standards and encryption protocols.
- Investigate best practices for securing email communication and protecting user data.
- Review case studies on successful implementations of secure email systems.

## 2.9 Testing and Quality Assurance in Web Development:

- Review literature on testing methodologies for web applications, including unit testing, integration testing, and user acceptance testing.
- Explore best practices for ensuring the quality and reliability of web applications.

## 2.10 Deployment and Monitoring:

- Investigate literature on best practices for deploying web applications to hosting environments.
- Explore studies on monitoring and logging strategies for web applications to ensure ongoing performance optimization

# CHAPTER 3

# FEASIBILITY STUDY

After doing the project, study and analyzing all the existing or required functionalities of the system, the next task is to do the feasibility study for the project. All projects are feasible-given unlimited resources and in finite time. Feasibility study includes consideration of all the possible ways to provide a solution to the given problem. The proposed solution should satisfy all the user requirements and should be flexible enough so that future changes can be easily done based on the future upcoming requirements. There are four parts in feasibility study

- Technical Feasibility
- Operational Feasibility
- Legal Feasibility
- Economical Feasibility
- Scheduling Feasibility

## 3.1 TECHNICAL FEASIBILITY

The technical feasibility involves evaluating the existing technologies and expertise required for the project. The chosen technologies—React.js for the frontend, Node.js with Express.js for the backend, and MongoDB for data storage—are widely used and supported. The availability of skilled developers and the potential need for training are considered in this assessment.

## 3.2 OPERATIONAL FEASIBILITY

Operational feasibility assesses the resources required and analyzes the workflow of email creation, sending, receiving, and organization. The project identifies necessary resources, such as hardware and software, and considers potential bottlenecks in the workflow to optimize operational efficiency. A detailed analysis of resource requirements is crucial for effective project planning and execution.

An in-depth analysis of the workflow for email creation, sending, receiving, and organization is conducted. This includes identifying potential bottlenecks and areas for improvement. Optimizing the workflow ensures operational efficiency and a seamless user experience.

## 3.3 LEGAL FEASIBILITY

The legal feasibility assessment ensures compliance with data protection laws, privacy regulations, and other legal requirements. A detailed review is conducted to confirm that the project adheres to relevant legal standards.
The project undergoes a thorough examination to confirm adherence to legal requirements, including the verification of intellectual property rights.

## 3.4 ECONOMICAL FEASIBILITY

Economic feasibility involves estimating the costs associated with the project and analyzing the potential return on investment (ROI). Cost estimates cover development, deployment, and ongoing maintenance. The ROI analysis considers the financial gains and benefits compared to the project's costs, ensuring economic viability.
Detailed financial projections are prepared, considering hardware and software costs, human resources, training expenses, and other associated costs.

## 3.5 SCHEDULING FEASIBILITY

Scheduling feasibility involves developing a realistic timeline for project completion. A phased approach is considered, including planning, development, testing, and deployment. Dependencies and potential risks that could impact the project schedule are identified, allowing for effective project management.

# CHAPTER 4

# REQUIREMENT ANALYSIS

The requirement analysis phase is a critical step in understanding and defining the functionalities, features, and constraints of the Gmail Clone project.

## 4.1 FUNCTIONAL REQUIREMENT

The Gmail Clone project's functional requirements include user registration and authentication, email composition with rich text formatting and attachments, organized email reception and viewing, customizable labeling for email organization, robust search functionality, contacts management with auto-suggestion, a responsive design for various device:

### 4.1.1 User Registration and Authentication:

- Users should be able to register for an account securely.
- Authentication mechanisms should be in place to protect user accounts.

### 4.1.2 Email Composition and Sending:

- Users should have the ability to compose emails with support for rich text formatting.
- Attachment functionality for sending files should be included.

### 4.1.3 Email Reception and Viewing:

- Users should be able to receive emails in a structured and organized manner.
- Email viewing should support threaded conversations.

### 4.1.4 Search Functionality:

- The system should provide robust search functionality to enable users to find specific emails quickly.

- Advanced search filters based on sender, subject, date, etc., should be available.

**4.1.5 Contacts Management:**

- Users should be able to manage a list of contacts, including adding, editing, and deleting contacts.

## 4.2 NON-FUNCTIONAL REQUIREMENT

The non-functional requirements for the Gmail Clone project encompass various aspects that contribute to the overall performance, usability, and security of the application. These include:

**4.2.1. Performance:**

- Response Time: Ensure low latency in loading emails and navigating through the application.
- Scalability: Support a large number of concurrent users with consistent performance.

**4.2.2. Usability:**

- User Interface Design: Provide an intuitive and user-friendly interface with modern design principles.
- Accessibility: Ensure the application is accessible to users with disabilities.

**4.2.3. Security:**

- Data Encryption: Implement end-to-end encryption for secure data transmission.

**4.2.4. Reliability:**

- System Uptime: Aim for high availability and minimize downtime for system maintenance.
- Error Handling: Implement effective error handling to provide users with meaningful error messages.

**4.2.5. Scalability:**

- Database Scalability: Design the database to scale horizontally to handle a growing volume of data.
- Load Balancing: Implement load balancing for even distribution of user requests.

**4.2.6. Compatibility:**

- Cross-Browser Compatibility: Ensure the application functions seamlessly across various web browsers.

- Device Compatibility: Support a diverse range of devices, including desktops, tablets, and mobile phones.

**4.2.7. Maintainability:**

- Code Maintainability: Follow coding best practices and modularize the code for ease of maintenance.

- Documentation: Maintain comprehensive documentation for codebase understanding and future development.

**4.2.8. User Experience:**

- Loading Time: Minimize loading times for emails, attachments, and other application components.

These non-functional requirements collectively contribute to the quality, reliability, and user satisfaction of the Gmail Clone project, ensuring that the application not only functions well but also provides a secure and enjoyable user experience.

## 4.3 SOFTWARE REQUIREMENT

**Table 4.1 Software Requirement for Gmail Clone**

| S. NO. | DESCRIPTION | TYPE |
|---|---|---|
| 1 | Operating System | Windows, MacOS |
| 2 | Language | HTML5, CSS3, JavaScript, Bootstrap, ExpressJS, NodeJS |
| 3 | IDE | VS Code |
| 4 | Database | MongoDB |

## 4.4 HARDWARE REQUIREMENTS

**Table 4.2 Hardware Requirement for Gmail Clone**

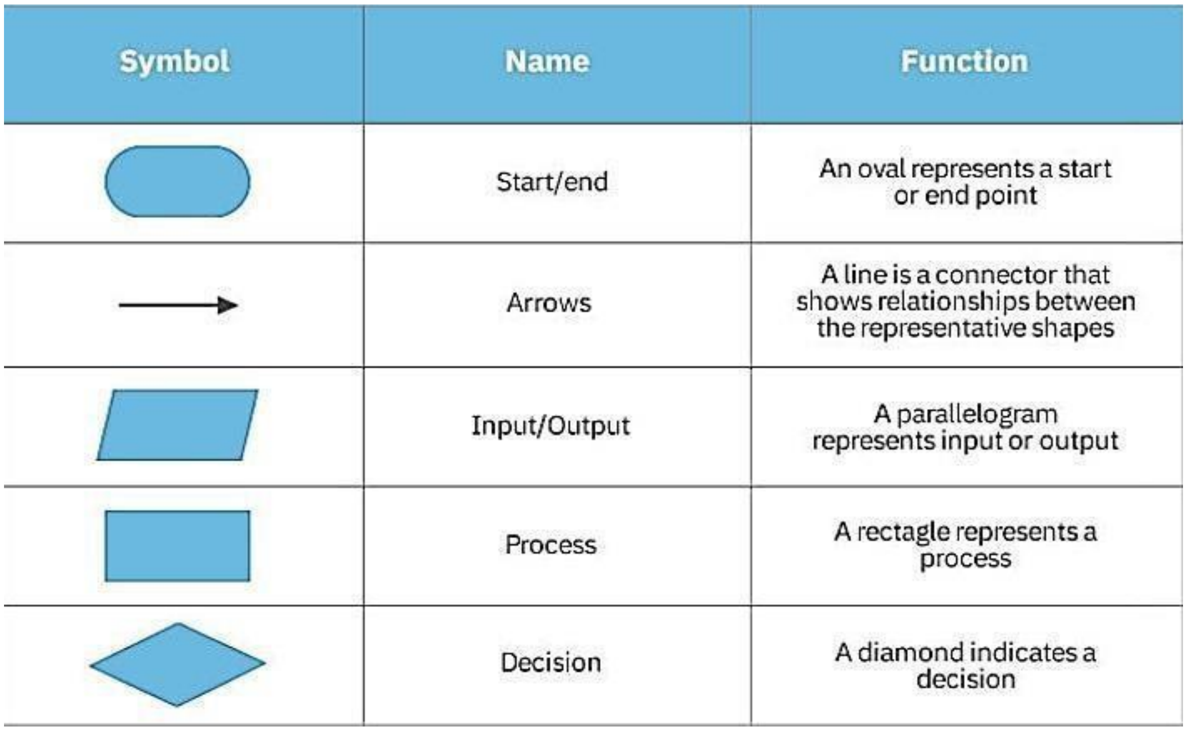
| S. NO. | DESCRIPTION | TYPE |
|---|---|---|
| 1 | Hardware | I3 Processor |
| 2 | Clock Speed | 3.0GHz |
| 3 | RAM | 8GB |
| 4 | SSD | 512GB |

# CHAPTER 5

# SYSTEM ARCHITECTURE AND DESIGN

## 5.1 FLOW CHART DIAGRAM

A flowchart is a visual representation of the sequence of steps and decisions needed to perform a process. Each step in the sequence is noted within a diagram shape. Steps are linked by connecting lines and directional arrows. This allows anyone to view the flowchart and logically follow the process from beginning to end. A flowchart is a powerful business tool. With proper design and construction it communicates the steps in a process very effectively and efficiently



| Symbol | Name | Function |
|---|---|---|
| ⬭ | Start/end | An oval represents a start or end point |
| → | Arrows | A line is a connector that shows relationships between the representative shapes |
| ▱ | Input/Output | A parallelogram represents input or output |
| ▭ | Process | A rectagle represents a process |
| ◆ | Decision | A diamond indicates a decision |

**Fig 5.1.1: Flow chart Symbols**

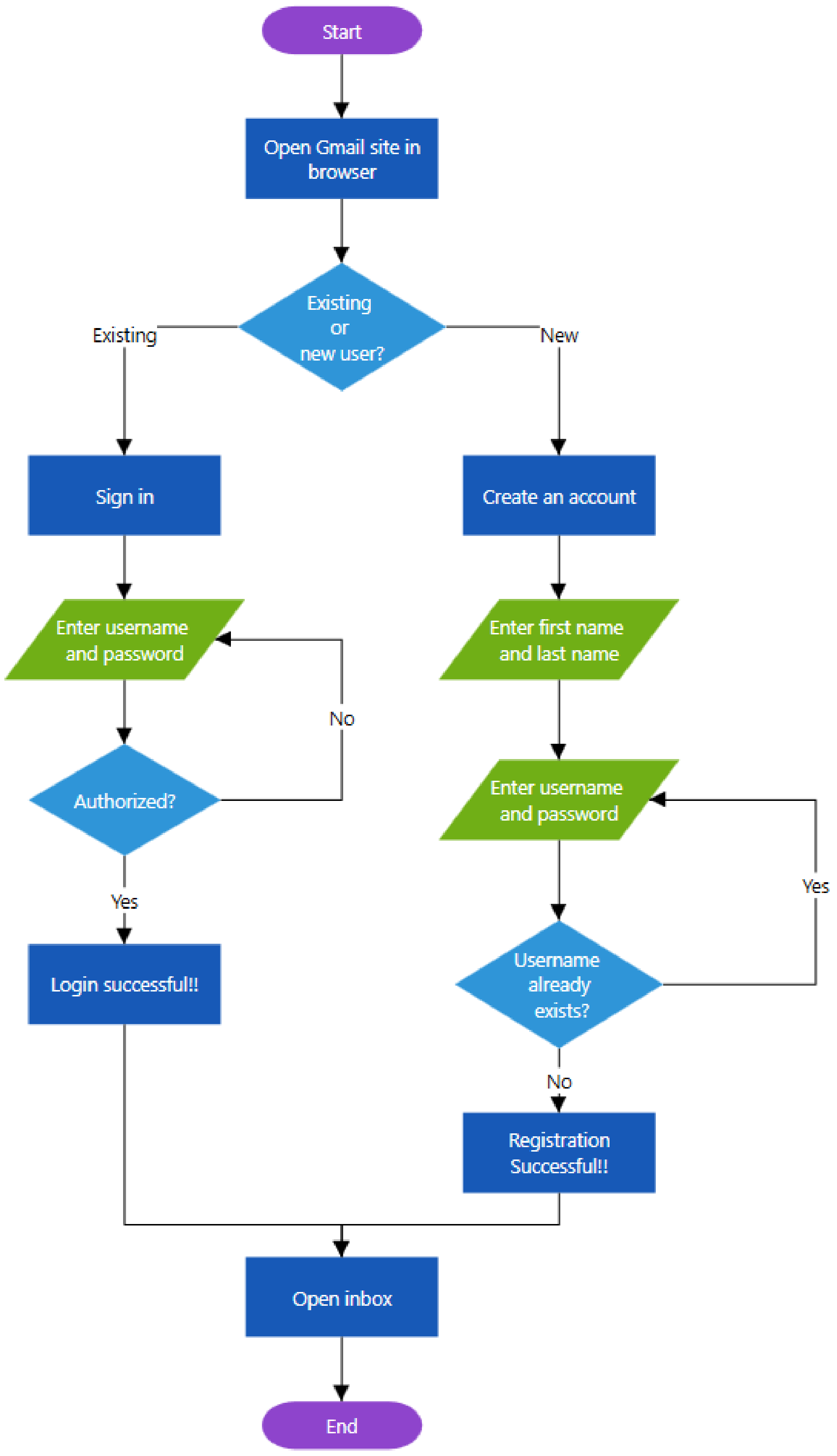


**Fig 5.1.2 Flow Chart Diagram**

## 5.2 ENTITY RELATIONSHIP DIAGRAM

Entity-Relationship model stands for an ER model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system. It develops a conceptual design for the database. It also develops a very simple and easy to design view of data. In ER modelling, the database structure is portrayed as a diagram called an entity relationship diagram

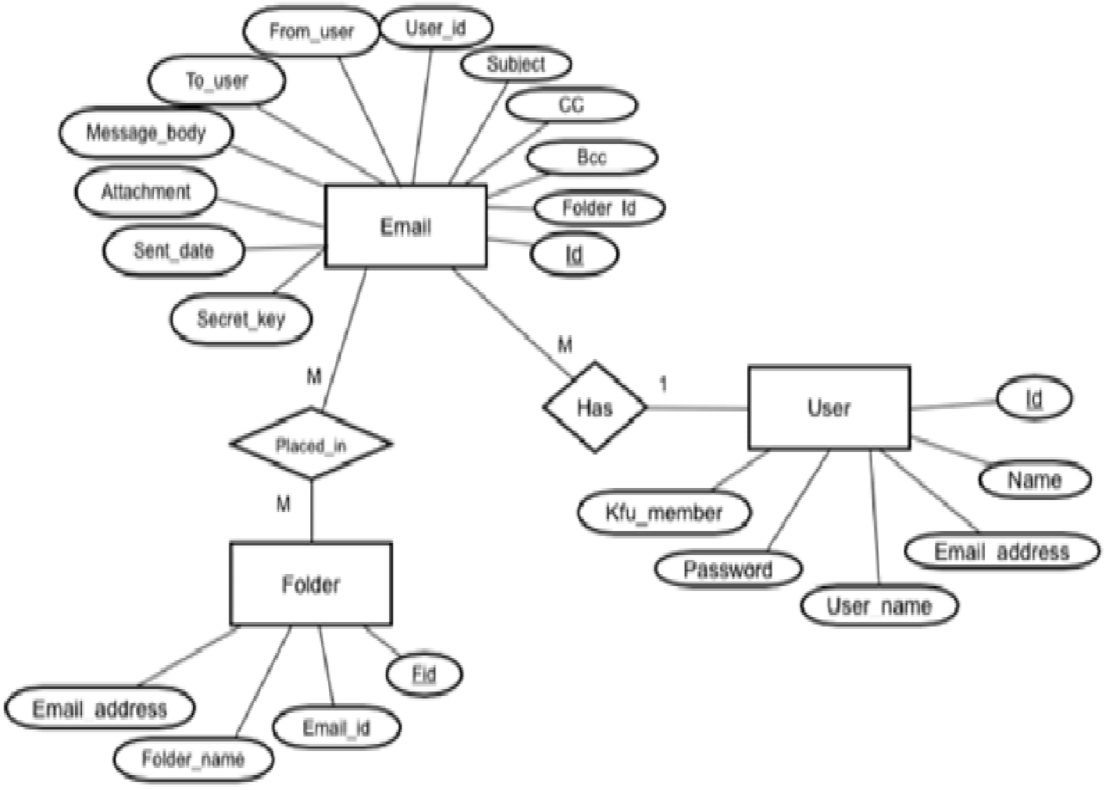


**Fig 5.2 Entity-Relationship Diagram**

## 5.3 USE CASE DIAGRAM

A use case diagram (UCD) is used to represent the dynamic behaviour of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.

The main purpose of a use case diagram is to portray the dynamic aspect of a system. It accumulates the system's requirement, which includes both internal as well as external influences. It invokes persons, use cases, and several things that invoke the actors and elements accountable for the implementation of use case diagrams. It represents how an entity from the external environment can interact with a part of the system.

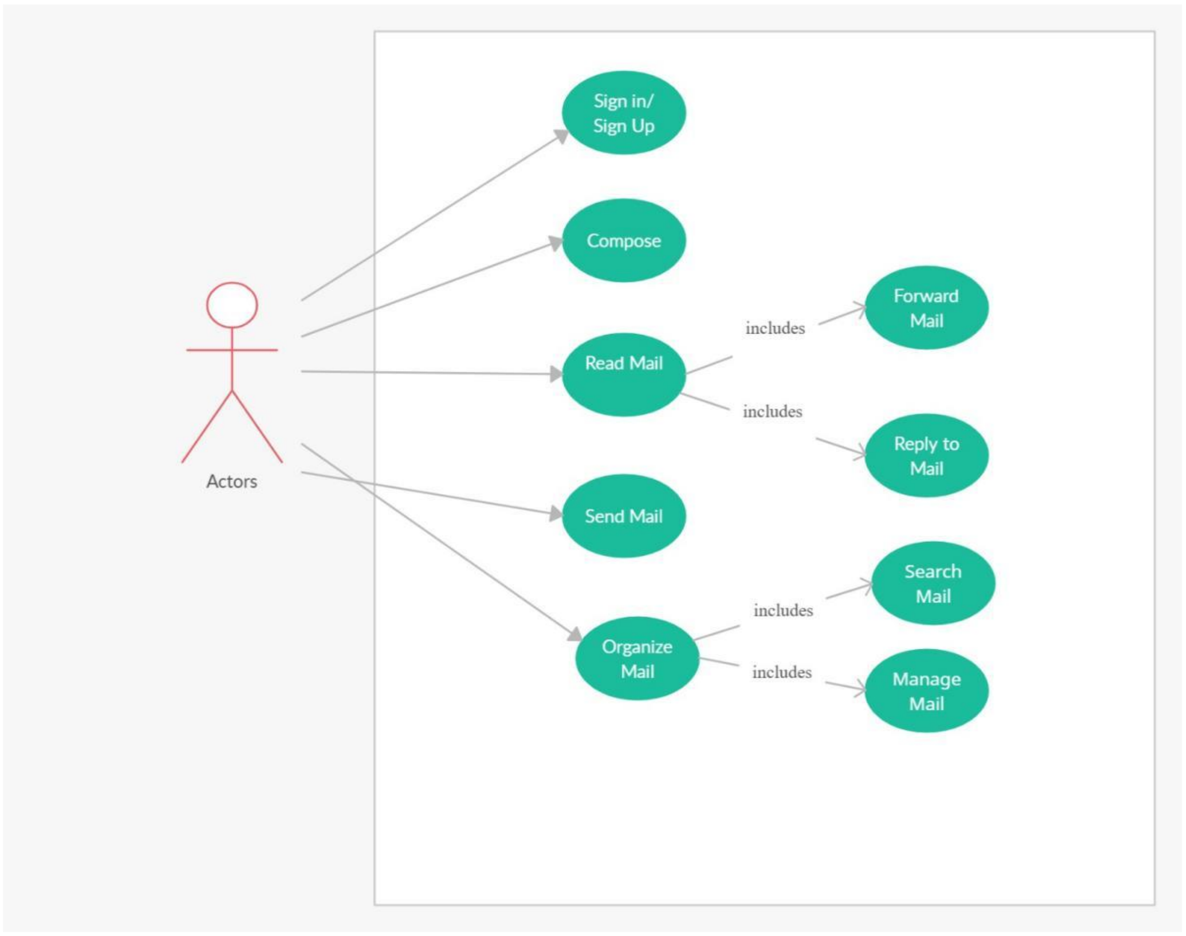


Fig 5.3 Use-Case Diagram

## 5.4 CLASS DIAGRAM

The class diagram for the Gmail Clone project illustrates the key classes and their relationships. It includes classes such as User, Email, Contact, and Label, each encapsulating relevant attributes and methods. Associations depict connections between classes, such as a User having multiple Emails. This visual representation serves as a blueprint for the project's object-oriented structure, fostering a clear understanding of class interactions and system architecture

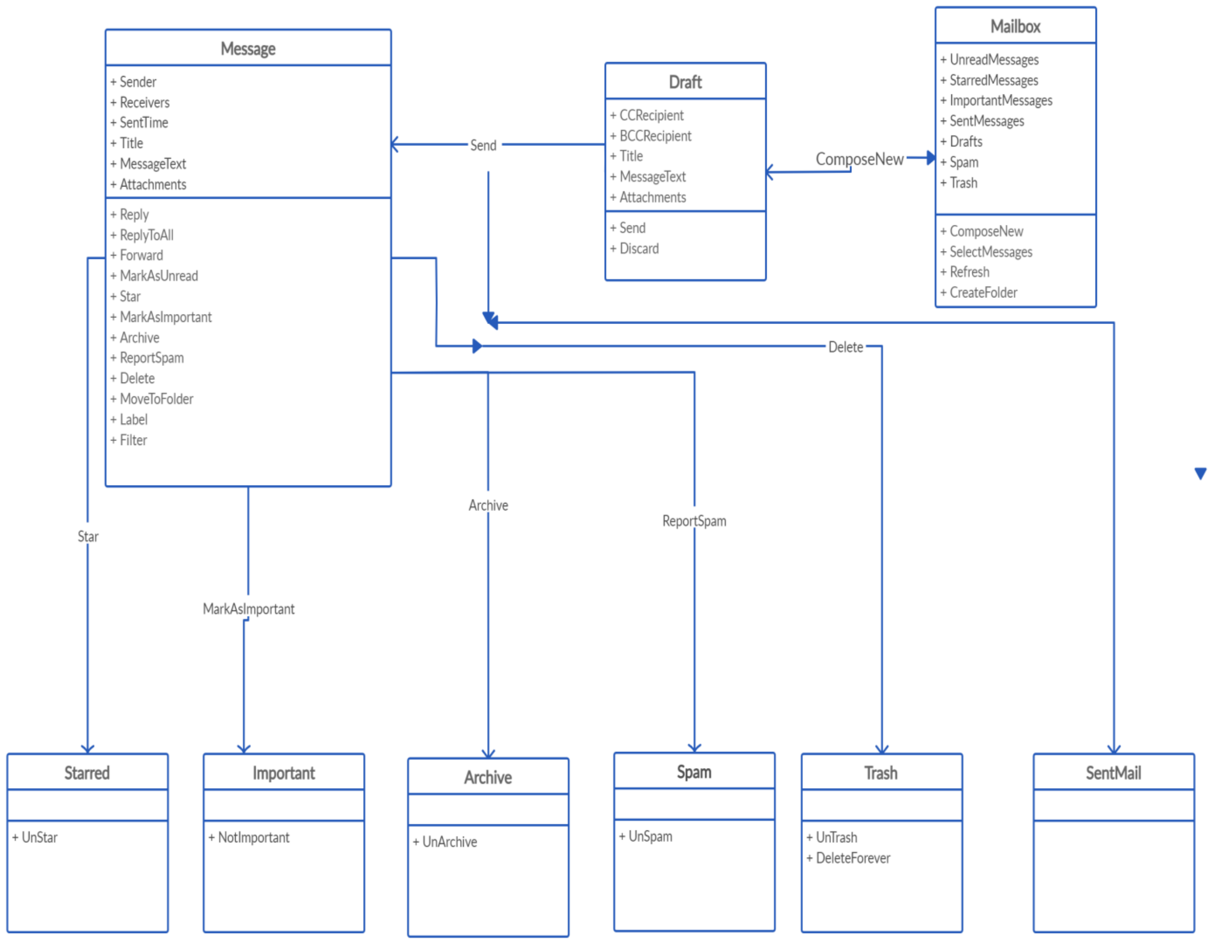


**Fig 5.4 Class Diagram**

# CHAPTER 6

# FORM DESIGN

## 6.1 SIGNIN PAGE

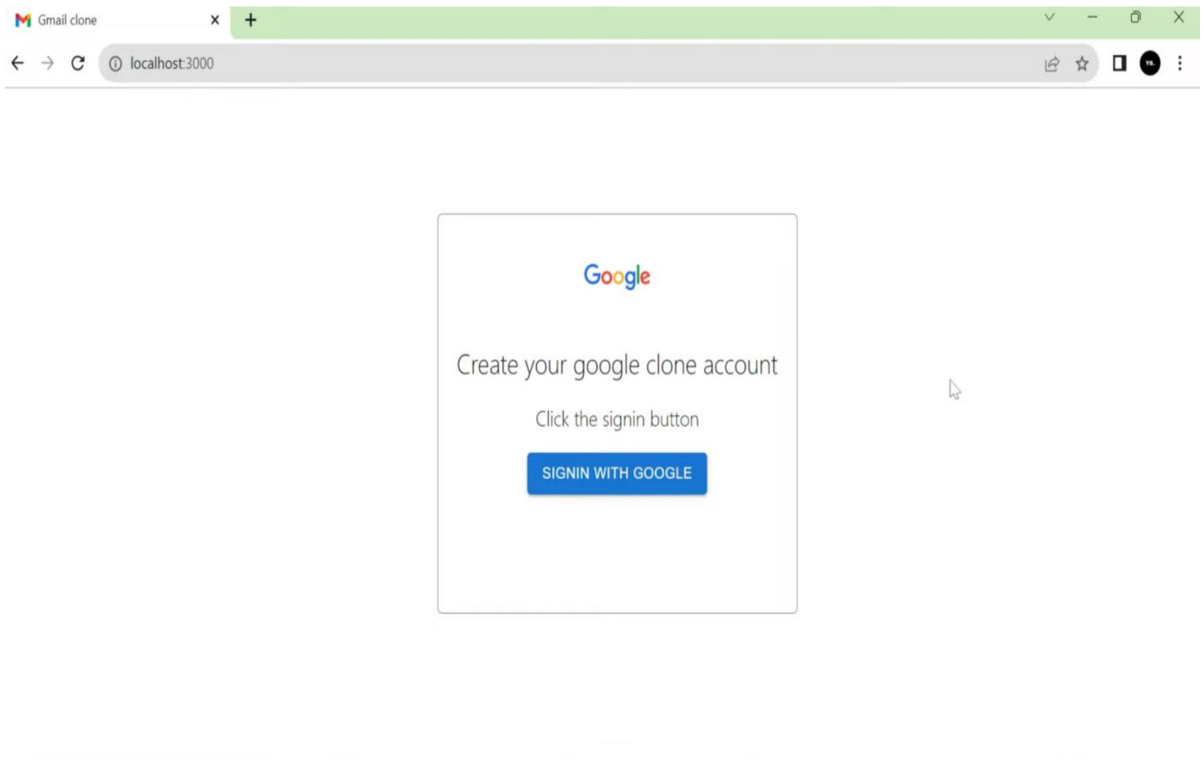


**Fig 6.1 Sign Page**

The Gmail Clone project's sign-in/sign-up page is designed for seamless user authentication. Users can securely sign in with existing credentials or register for a new account. The page prioritizes a user-friendly interface, robust password policies, and adherence to security best practices, ensuring a secure and efficient authentication process.

## 6.2 Inbox

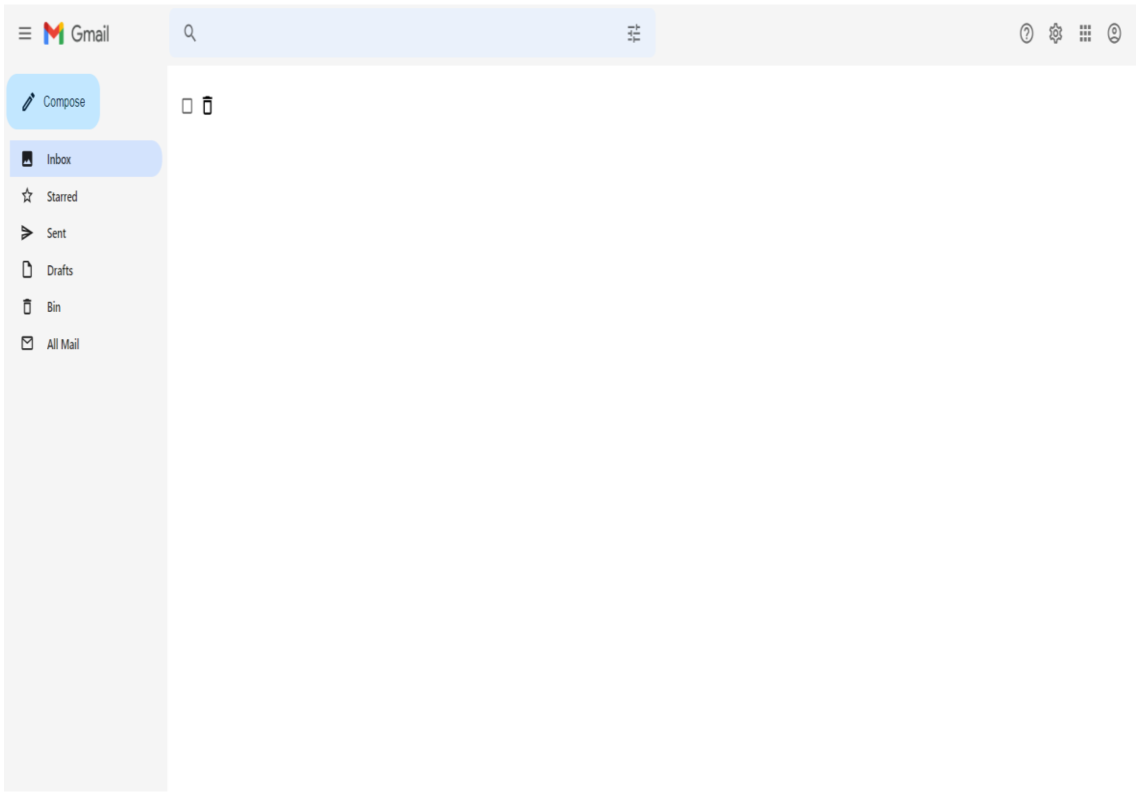
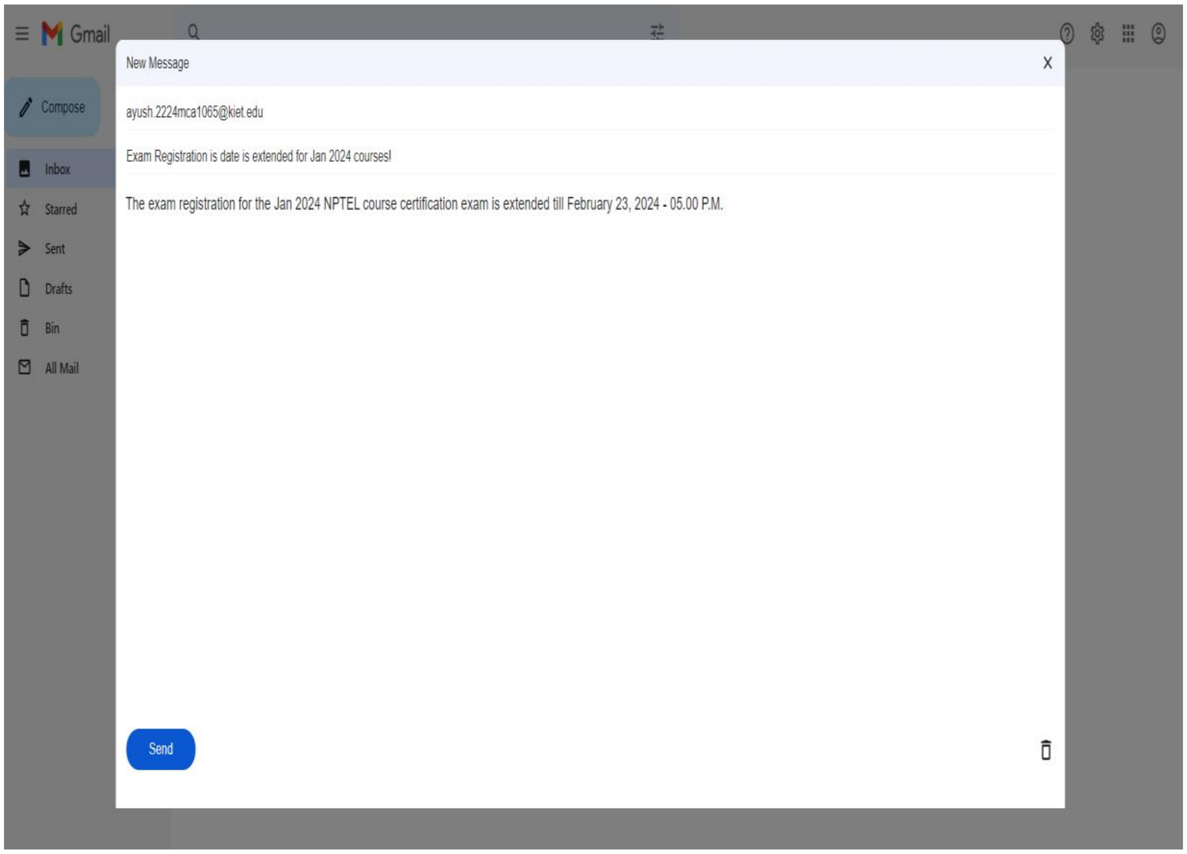


**Fig 6.2 Inbox**

The Gmail Clone project's inbox offers a user-friendly interface for managing emails. Users can efficiently navigate through received emails, view threaded conversations, and organize messages using customizable labels. Robust search functionality enables quick access to specific emails, enhancing user convenience. The responsive design ensures a consistent experience across devices. Advanced features include attachment support, inline images, and intuitive organization, creating a modern and efficient email management platform.

## 6.3 Compose



ayush.2224mca1065@kiet.edu

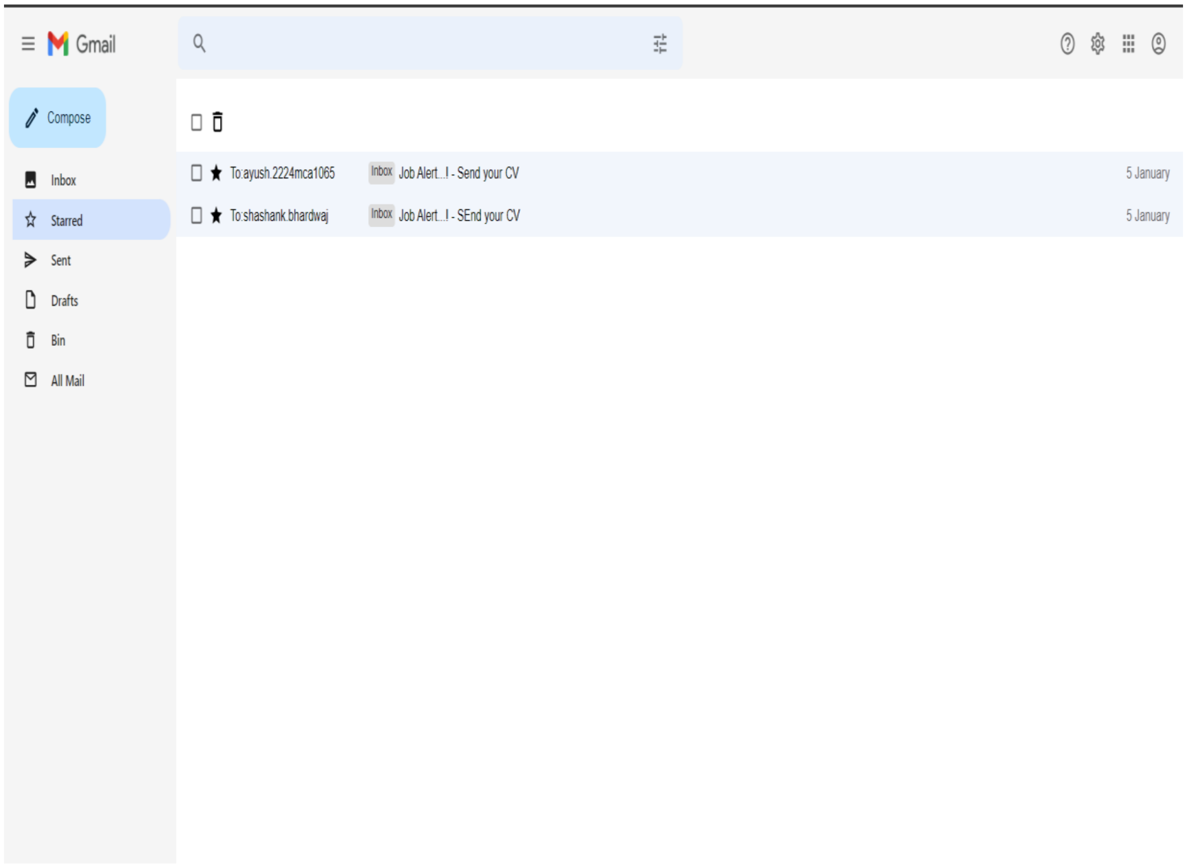
Exam Registration is date is extended for Jan 2024 courses!

The exam registration for the Jan 2024 NPTEL course certification exam is extended till February 23, 2024 - 05.00 P.M.

**Fig 6.3 Compose**

The Gmail Clone's compose mail feature offers a user-friendly interface for creating and sending emails. Users can efficiently draft messages with support for rich text formatting, allowing for bold text, italics, and more. The feature includes seamless attachment functionality, enabling users to include files and documents with ease. The intuitive design facilitates a smooth email composition process, ensuring a hassle-free experience. Additionally, inline image support enhances the visual appeal of emails. The feature-rich compose mail functionality aligns with modern email standards, providing users with a versatile and comprehensive tool for effective communication within the Gmail Clone platform.

## 6.4 Starred Message

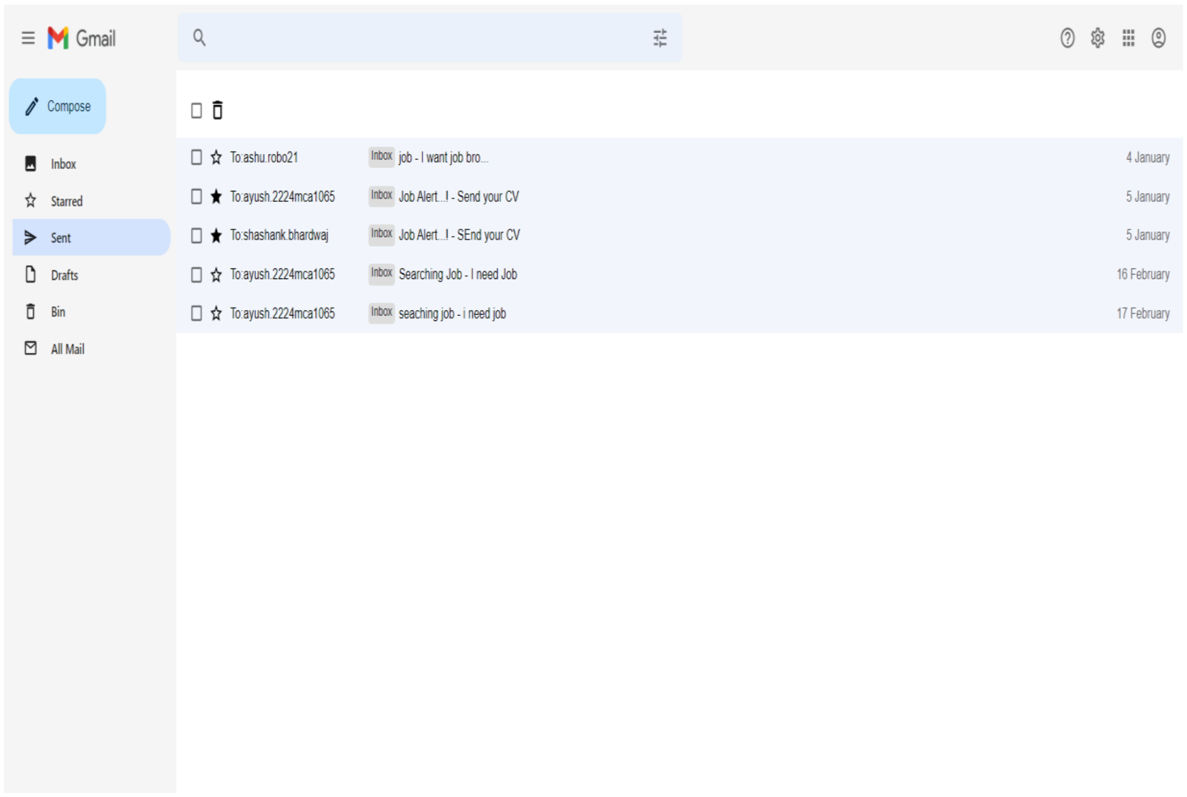


To:ayush.2224mca1065 Inbox Job Alert...! - Send your CV 5 January
To:shashank.bhardwaj Inbox Job Alert...! - SEnd your CV 5 January

**Fig 6.4 Starred Message**

The Gmail Clone project incorporates a robust "Starred Messages" feature that allows users to highlight and prioritize important emails. Users can easily mark messages with a star, serving as a visual indicator for crucial correspondence. This feature aids in efficient email organization, making it simple for users to identify and access their most significant messages. Starred messages are conveniently accessible in a dedicated folder or view, streamlining the process of managing and retrieving important information. This functionality enhances user productivity by providing a quick and effective method for flagging and retrieving key messages, contributing to an organized and streamlined email management experience within the Gmail Clone application.

## 6.5 Send Mail

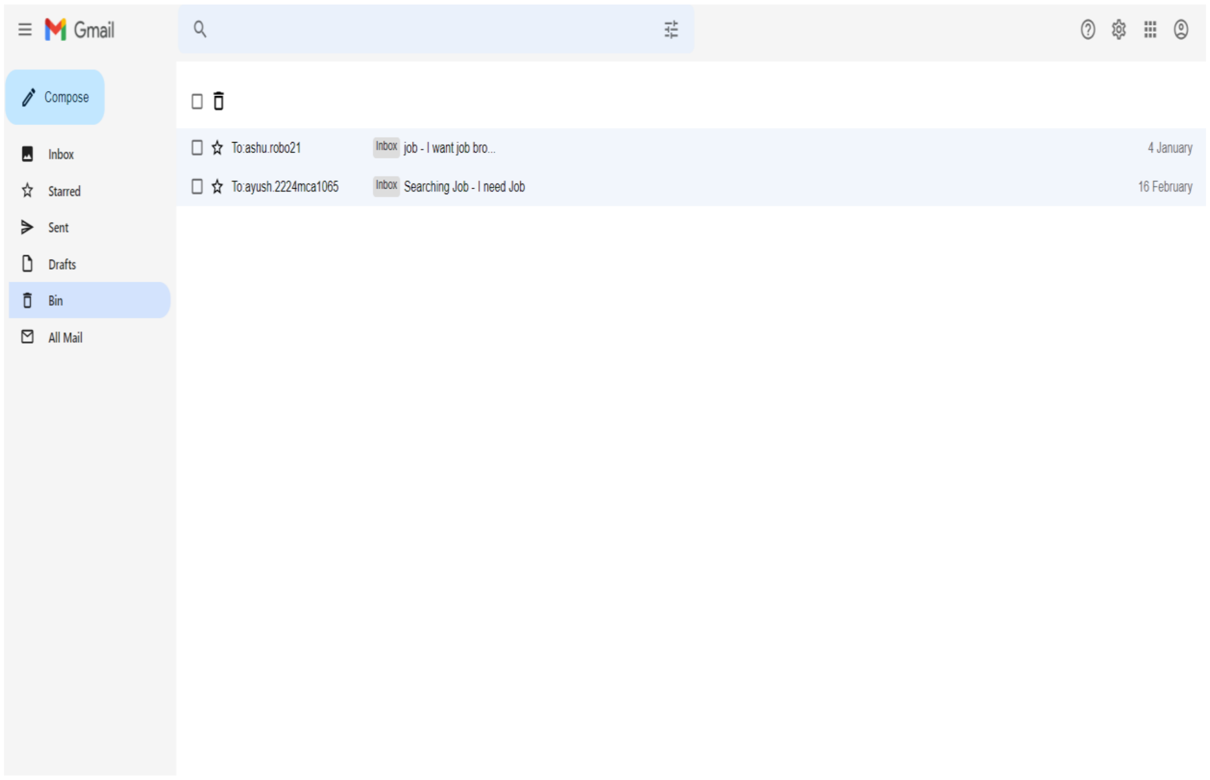


To:ashu.robo21 Inbox job - I want job bro... 4 January
To:ayush.2224mca1065 Inbox Job Alert...! - Send your CV 5 January
To:shashank.bhardwaj Inbox Job Alert...! - SEnd your CV 5 January
To:ayush.2224mca1065 Inbox Searching Job - I need Job 16 February
To:ayush.2224mca1065 Inbox seaching job - i need job 17 February

**Fig 6.5 Send Mail**

The "Send Mail" functionality in the Gmail Clone project offers a seamless and efficient way for users to dispatch emails. Users can compose messages with rich text formatting, ensuring clarity and customization. The feature includes attachment support, enabling users to include files or documents with ease. Once the email is crafted, users can select recipients from their contact list, and the system provides auto-suggestions for efficiency. The intuitive design of the "Send Mail" functionality ensures a smooth user experience, and users can review their composed email before hitting the send button. This feature also incorporates inline image support, allowing users to include visuals directly within their messages. With robust security measures in place, the "Send Mail" functionality ensures a secure and user-friendly communication experience within the Gmail Clone platform.

## 6.6 Bin



To:ashu.robo21 Inbox job - I want job bro... 4 January
To:ayush.2224mca1065 Inbox Searching Job - I need Job 16 February

**Fig 6.6 Bin**

The Gmail Clone project integrates a comprehensive "Bin" feature, providing users with an effective method for managing deleted emails. Deleted emails are temporarily moved to the Bin, allowing users the option to restore or permanently delete them. This feature prevents accidental loss of important information and facilitates a user-friendly recovery process. The Bin is equipped with advanced search capabilities, enabling users to locate specific deleted emails efficiently. Additionally, the Bin supports bulk actions for streamlined management, enhancing user control over discarded content. The Gmail Clone's Bin feature aligns with modern email management practices, offering a balance between user convenience and data control. Its functionality ensures that users can easily retrieve mistakenly deleted emails or confidently remove unnecessary content, contributing to an organized and efficient email management experience within the application.

# CHAPTER 7
# CODING

## Email.jsx

```
import { ListItem, Checkbox, Typography, Box, styled } from "@mui/material";
import { StarBorder, Star } from '@mui/icons-material';
import useApi from '../hooks/useApi';
import { API_URLS } from "../services/api.urls";
import { useNavigate } from "react-router-dom";
import { routes } from "../routes/routes";

const Wrapper = styled(ListItem)`
    padding: 0 0 0 10px;
    background: #f2f6fc;
    cursor: pointer;
    & > div {
        display: flex;
        width: 100%
    }
    & > div > p {
        font-size: 14px;
    }
```

```
`;

const Indicator = styled(Typography)`
    font-size: 12px !important;
    background: #ddd;
    color: #222;
    border-radius: 4px;
    margin-right: 6px;
    padding: 0 4px;
`;

const Date = styled(Typography)({
    marginLeft: 'auto',
    marginRight: 20,
    fontSize: 12,
    color: '#5F6368'
})

const Email = ({ email, setStarredEmail, selectedEmails, setSelectedEmails }) => {
    const toggleStarredEmailService = useApi(API_URLS.toggleStarredMails);

    const navigate = useNavigate();

    const toggleStarredEmail = () => {
        toggleStarredEmailService.call({ id: email._id, value: !email.starred });
        setStarredEmail(prevState => !prevState);
    }
    const handleChange = () => {
        if (selectedEmails.includes(email._id)) {
            setSelectedEmails(prevState => prevState.filter(id => id !== email._id));
        } else {
            setSelectedEmails(prevState => [...prevState, email._id]);
        }
    }

    return (
        <Wrapper>
            <Checkbox
                size="small"
                checked={selectedEmails.includes(email._id)}
                onChange={() => handleChange()}
            />
            {
                email.starred ?
                    <Star fontSize="small" style={{ marginRight: 10 }} onClick={() =>
toggleStarredEmail()} />
                :
                    <StarBorder fontSize="small" style={{ marginRight: 10 }} onClick={() =>
toggleStarredEmail()} />
            }
            <Box onClick={() => navigate(routes.view.path, { state: { email: email }})}>
                <Typography style={{ width: 200 }}>To:{email.to.split('@')[0]}</Typography>
                <Indicator>Inbox</Indicator>
                <Typography>{email.subject} {email.body && '-'} {email.body}</Typography>
                <Date>
                    {(new window.Date(email.date)).getDate()}&nbsp;
                    {(new window.Date(email.date)).toLocaleString('default', { month: 'long' })}
                </Date>
            </Box>
        </Wrapper>
    )
}

export default Email;
```

## index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

```
reportWebVitals();
```

## ComposeMail.jsx

```
import { useState } from 'react';


import { Dialog, styled, Typography, Box, InputBase, TextField, Button } from
'@mui/material';

import { Close, DeleteOutline } from '@mui/icons-material';

import useApi from '../hooks/useApi';

import { API_URLS } from '../services/api.urls';


const dialogStyle = {

   height: '90%',

   width: '80%',

   maxWidth: '100%',

   maxHeight: '100%',

   boxShadow: 'none',

   borderRadius: '10px 10px 0 0',

}


const Header = styled(Box)`

   display: flex;

   justify-content: space-between;

   padding: 10px 15px;

   background: #f2f6fc;

   & > p {
       font-size: 14px;
       font-weight: 500;
   }
`;

const RecipientWrapper = styled(Box)`
   display: flex;
   flex-direction: column;
   padding: 0 15px;
   & > div {
       font-size: 14px;
       border-bottom: 1px solid #F5F5F5;
       margin-top: 10px;
   }
`;

const Footer = styled(Box)`
   display: flex;
   justify-content: space-between;
   padding: 10px 15px;
   align-items: center;
`;

const SendButton = styled(Button)`
   background: #0B57D0;
   color: #fff;
   font-weight: 500;
   text-transform: none;
   border-radius: 18px;
   width: 100px;
`

const ComposeMail = ({ open, setOpenDrawer }) => {
   const [data, setData] = useState({});
   const sentEmailService = useApi(API_URLS.saveSentEmails);
   const saveDraftService = useApi(API_URLS.saveDraftEmails);

   const config = {
       Username: process.env.REACT_APP_USERNAME,
       Password: process.env.REACT_APP_PASSWORD,
       Host: 'smtp.elasticemail.com',
       Port: 2525,
   }

   const onValueChange = (e) => {
       setData({ ...data, [e.target.name]: e.target.value })
   }

   const sendEmail = async (e) => {
       e.preventDefault();

       if (window.Email) {
           window.Email.send({
               ...config,
               To : data.to,
               From : "codeforinterview03@gmail.com",
               Subject : data.subject,
               Body : data.body
           }).then(
               message => alert(message)
           );
       }
1
       const payload = {
           to : data.to,
           from : "codeforinterview03@gmail.com",
           subject : data.subject,
           body : data.body,
           date: new Date(),
           image: '',
           name: 'Code for Interview',
           starred: false,
           type: 'sent'
       }

       sentEmailService.call(payload);

       if (!sentEmailService.error) {
           setOpenDrawer(false);
           setData({});
       } else {
       }
   }

   const closeComposeMail = (e) => {
       e.preventDefault();

       const payload = {
           to : data.to,
           from : "codeforinterview03@gmail.com",
           subject : data.subject,
           body : data.body,
           date: new Date(),
           image: '',
           name: 'Code for Interview',
           starred: false,
           type: 'drafts'
       }

       saveDraftService.call(payload);

       if (!saveDraftService.error) {
           setOpenDrawer(false);
           setData({});
       } else {

       }
```

```
   }


   return (
      <Dialog
         open={open}
         PaperProps={{ sx: dialogStyle }}
      >
         <Header>
            <Typography>New Message</Typography>
            <Close fontSize="small" onClick={(e) => closeComposeMail(e)} />
         </Header>
         <RecipientWrapper>
            <InputBase placeholder='Recipients' name="to" onChange={(e) =>
onValueChange(e)} value={data.to} />
            <InputBase placeholder='Subject' name="subject" onChange={(e) =>
onValueChange(e)} value={data.subject} />
         </RecipientWrapper>
         <TextField
            multiline
            rows={20}
            sx={{ '& .MuiOutlinedInput-notchedOutline': { border: 'none' } }}
            name="body"
            onChange={(e) => onValueChange(e)}
            value={data.body}
         />
         <Footer>
            <SendButton onClick={(e) => sendEmail(e)}>Send</SendButton>
            <DeleteOutline onClick={() => setOpenDrawer(false)} />
            </Footer>
        </Dialog>
    )
}

export default ComposeMail;
```

## Header.jsx

```
import { AppBar, Toolbar, Box, InputBase, styled } from '@mui/material';
import { Menu as MenuIcon, Tune, HelpOutlineOutlined, SettingsOutlined,
    AppsOutlined, AccountCircleOutlined, Search } from '@mui/icons-material'

import { gmailLogo } from '../constants/constant';

const StyledAppBar = styled(AppBar)`
    background: #f5F5F5;
    box-shadow: none;
`;

const SearchWrapper = styled(Box)`
    background: #EAF1FB;
    margin-left: 80px;
    border-radius: 8px;
    min-width: 690px;
    max-width: 720px;
    height: 48px;
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 0 20px;
    & > div {
        width: 100%
    }
`

const OptionsWrapper = styled(Box)`
    width: 100%;
    display: flex;
    justify-content: end;
    & > svg {
        margin-left: 20px;
    }
`

const Header = ({ toggleDrawer }) => {

    return (
        <StyledAppBar position="static">
            <Toolbar>
                <MenuIcon color="action" onClick={toggleDrawer} />
                <img src={gmailLogo} alt="logo" style={{ width: 110, marginLeft: 15 }} />
                <SearchWrapper>
                    <Search color="action" />
                    <InputBase />
```

```
          <Tune  color="action"/>
        </SearchWrapper>

        <OptionsWrapper>
          <HelpOutlineOutlined color="action" />
          <SettingsOutlined color="action" />
          <AppsOutlined color="action" />
          <AccountCircleOutlined color="action" />
       </OptionsWrapper>
     </Toolbar>
   </StyledAppBar>
 )
}

export default Header;
```

## Emails.jsx

```
import { useEffect, useState } from 'react';
import { useParams, useOutletContext } from 'react-router-dom';
import useApi from '../hooks/useApi';
import { API_URLS } from '../services/api.urls';
import { Box, List, Checkbox } from '@mui/material';
import Email from './Email';
import { DeleteOutline } from '@mui/icons-material';
import NoMails from './common/NoMails';
import { EMPTY_TABS } from '../constants/constant';
```

```
const Emails = () => {
    const [starredEmail, setStarredEmail] = useState(false);
    const [selectedEmails, setSelectedEmails] = useState([]);

    const { openDrawer } = useOutletContext();
    const { type } = useParams();

    const getEmailsService = useApi(API_URLS.getEmailFromType);
    const deleteEmailsService = useApi(API_URLS.deleteEmails);
    const moveEmailsToBin = useApi(API_URLS.moveEmailsToBin);

    useEffect(() => {
        getEmailsService.call({}, type);
    }, [type, starredEmail])

    const selectAllEmails = (e) => {
        if (e.target.checked) {
            const emails = getEmailsService?.response?.map(email => email._id);
            setSelectedEmails(emails);
        } else {
            setSelectedEmails([]);
        }
    }

    const deleteSelectedEmails = () => {
        if (type === 'bin') {
            deleteEmailsService.call(selectedEmails);
        } else {
            moveEmailsToBin.call(selectedEmails);
        }
        setStarredEmail(prevState => !prevState);
    }

    return (
        <Box style={openDrawer ? { marginLeft: 250, width: '100%' } : { width: '100%' } }>
            <Box style={{ padding: '20px 10px 0 10px', display: 'flex', alignItems: 'center' }}>
                <Checkbox size="small" onChange={(e) => selectAllEmails(e)} />
                <DeleteOutline onClick={(e) => deleteSelectedEmails(e)} />
            </Box>
            <List>
                {
                    getEmailsService?.response?.map(email => (
                        <Email
                            email={email}
                            key={email.id}
                            setStarredEmail={setStarredEmail}
                            selectedEmails={selectedEmails}
                            setSelectedEmails={setSelectedEmails}
                        />
                    ))
                }
            </List>
            {
                getEmailsService?.response?.length === 0 &&
                <NoMails message={EMPTY_TABS[type]} />
            }
        </Box>
    )
}

export default Emails;
```

**SideBar.jsx**

```
import { Drawer, styled } from "@mui/material";

import SideBarContent from "./SideBarContent";

const StyledDrawer = styled(Drawer)`
    margin-top: 54px;
`

const SideBar = ({ toggleDrawer, openDrawer }) => {

    return (
        <StyledDrawer
            anchor='left'
            open={openDrawer}
            onClose={toggleDrawer}
            hideBackdrop={true}
            ModalProps={{
                keepMounted: true,
            }}
            variant="persistent"
            sx={{
                '& .MuiDrawer-paper': {
                    width: 250,
                    borderRight: 'none',
                    background: '#f5F5F5',
                    marginTop: '64px',
                    height: 'calc(100vh - 64px)'
                },
            }}
          >
            <SideBarContent />
        </StyledDrawer>
    )
}

export default SideBar;
```

## SideBarContent.jsx

```
import { useState } from 'react';
import { Button, List, ListItem, Box, styled } from '@mui/material';
import ComposeMail from './ComposeMail';
import { SIDEBAR_DATA } from '../config/sidebar.config';
import { CreateOutlined } from '@mui/icons-material';
import { NavLink, useParams } from 'react-router-dom';
import { routes } from '../routes/routes';

const Container = styled(Box)`
  padding: 8px;
  & > ul {
    padding: 10px 0 0 5px;
    font-size: 14px;
    font-weight: 500;
    cursor: pointer;
    & > a {
      text-decoration: none;
      color: inherit;
    }
    & > a > li > svg {
      margin-right: 20px;
    }
  }
`

const ComposeButton = styled(Button)`
  background: #c2e7ff;
  color: #001d35;
  border-radius: 16px;
  padding: 15px;
  min-width: 140px;
  text-transform: none;
`
```

```
}



const SideBarContent = () => {

    const [openDrawer, setOpenDrawer] = useState(false);

    const { type } = useParams();

    const onComposeClick = () => {
        setOpenDrawer(true);
    }

    return (
        <Container>
            <ComposeButton onClick={() => onComposeClick()}>
                <CreateOutlined style={{ marginRight: 10 }} />Compose
            </ComposeButton>
            <List>
                {
                    SIDEBAR_DATA.map(data => (
                        <NavLink key={data.name} to={`${routes.emails.path}/${data.name}`}>
                            <ListItem style={ type === data.name.toLowerCase() ? {
                                backgroundColor: '#d3e3fd',
                                borderRadius: '0 16px 16px 0'
                            } : {}}><data.icon fontSize="small" />{data.title}</ListItem>
                        </NavLink>
                    ))
                }
            </List>
            <ComposeMail open={openDrawer} setOpenDrawer={setOpenDrawer} />
        </Container>
    )
}

export default SideBarContent;
```

## ViewEmail.jsx

```
import { useState } from 'react';
import { Button, List, ListItem, Box, styled } from '@mui/material';
import ComposeMail from './ComposeMail';
import { SIDEBAR_DATA } from '../config/sidebar.config';
import { CreateOutlined } from '@mui/icons-material';
import { NavLink, useParams } from 'react-router-dom';
import { routes } from '../routes/routes';

const Container = styled(Box)`
    padding: 8px;
    & > ul {
        padding: 10px 0 0 5px;
        font-size: 14px;
        font-weight: 500;
        cursor: pointer;
        & > a {
            text-decoration: none;
            color: inherit;
        }
        & > a > li > svg {
            margin-right: 20px;
        }
    }
`

const ComposeButton = styled(Button)`
    background: #c2e7ff;
    color: #001d35;
    border-radius: 16px;
    padding: 15px;
    min-width: 140px;
    text-transform: none;
`

const SideBarContent = () => {

    const [openDrawer, setOpenDrawer] = useState(false);

    const { type } = useParams();

    const onComposeClick = () => {
        setOpenDrawer(true);
    }

    return (
        <Container>
            <ComposeButton onClick={() => onComposeClick()}>
                <CreateOutlined style={{ marginRight: 10 }} />Compose
            </ComposeButton>
            <List>
                {
                    SIDEBAR_DATA.map(data => (
                        <NavLink key={data.name} to={`${routes.emails.path}/${data.name}`}>
                            <ListItem style={ type === data.name.toLowerCase() ? {
                                backgroundColor: '#d3e3fd',
                                borderRadius: '0 16px 16px 0'
                            } : {}}><data.icon fontSize="small" />{data.title}</ListItem>
                        </NavLink>
                    ))
                }
            </List>
            <ComposeMail open={openDrawer} setOpenDrawer={setOpenDrawer} />
        </Container>
    )
}

export default SideBarContent;
```

## useApi.jsx

```
import { useState } from 'react';
import API from '../services/api';

const useApi = (urlObject) => {
  const [response, setResponse] = useState(null);
  const [error, setError] = useState("");
  const [isLoading, setIsLoading] = useState(false);

  const call = async (payload, type = '') => {
    setResponse(null);
    setIsLoading(true);
    setError("");

    try {
      let res = await API(urlObject, payload, type);
      setResponse(res.data);
    } catch (error) {
      setError(error.message);
    } finally {
      setIsLoading(false);
    }
  }

  return { call, response, error, isLoading };
}

export default useApi;
```

## routes.js

```
import { lazy } from 'react';

const Main = lazy(() => import('../pages/Main'));
const Emails = lazy(() => import('../components/Emails'));
const ViewEmail = lazy(() => import('../components/ViewEmail'));

const routes = {
   main: {
      path: '/',
      element: Main
   },
   emails: {
      path: '/emails',
      element: Emails
   },
   invalid: {
      path: '/*',
      element: Emails
   },
   view: {
      path: '/view',
      element: ViewEmail
   }
}

export { routes };
```

## DataProvider.jsx

```
import { createContext, useState } from 'react';
import { VIEWS } from '../constants/constant';
```

```
export const DataContext = createContext(null);

const DataProvider = ({ children }) => {
    const [view, setView] = useState(VIEWS.inbox);

    return (
        <DataContext.Provider value={{
            view,
            setView
        }}>
            {children}
        </DataContext.Provider>
    )
}

export default DataProvider;
```

## Main.jsx

```
import { useState, Suspense } from 'react';

import { Header, SideBar } from '../components';
import { Box, styled } from '@mui/material';
import { Outlet } from 'react-router-dom';
import SuspenseLoader from '../components/common/SuspenseLoader';

const Wrapper = styled(Box)`
    display: flex;
`;
```

```
const Main = () => {

    const [openDrawer, setOpenDrawer] = useState(true);

    const toggleDrawer = () => {
        setOpenDrawer(prevState => !prevState);
    }

    return (
        <>
            <Header toggleDrawer={toggleDrawer} />
            <Wrapper>
                <SideBar toggleDrawer={toggleDrawer} openDrawer={openDrawer} />
                <Suspense fallback={<SuspenseLoader />} >
                    <Outlet context={{ openDrawer }} />
                </Suspense>
            </Wrapper>
        </>
    )
}

export default Main;
```

## Api.js

```
import axios from 'axios';

const API_URI = 'http://localhost:8000'
const API_GMAIL = async (serviceUrlObject, requestData = {}, type) => {
    const { params, urlParams, ...body } = requestData;


    return await axios({
        method: serviceUrlObject.method,
        url: `${API_URI}/${serviceUrlObject.endpoint}/${type}`,
        data: requestData
    })
}


export default API_GMAIL;
```

## App.js

```
import { Suspense, lazy } from 'react';
import { Navigate, Route, createBrowserRouter, createRoutesFromElements,
RouterProvider } from 'react-router-dom';
import { routes } from "./routes/routes";
import SuspenseLoader from './components/common/SuspenseLoader';
import DataProvider from './context/DataProvider';


const ErrorComponent = lazy(() => import('./components/common/ErrorComponent'));


const router = createBrowserRouter(
  createRoutesFromElements(
    <Route>
      <Route path={routes.main.path} element={<Navigate
to={`${routes.emails.path}/inbox`} />} />
      <Route path={routes.main.path} element={<routes.main.element />} >
```

```
        <Route path={`${routes.emails.path}/:type`} element={<routes.emails.element />}
errorElement={<ErrorComponent />} />

        <Route path={routes.view.path} element={<routes.view.element />}
errorElement={<ErrorComponent />} />

      </Route>


      <Route path={routes.invalid.path} element={<Navigate
to={`${routes.emails.path}/inbox`} />} />

    </Route>

  )

)


function App() {

  return (

    <Suspense fallback={<SuspenseLoader />}>

      <DataProvider>

        <RouterProvider router={router} />

      </DataProvider>

    </Suspense>

  );

}


export default App;
```

## App.css

```
.App {

  text-align: center;

}
```

```
.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

## Index.css

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

## package.json

```
{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
```

```
  "@emotion/react": "^11.10.5",
  "@emotion/styled": "^11.10.5",
  "@mui/icons-material": "^5.11.0",
  "@mui/material": "^5.11.7",
  "@testing-library/jest-dom": "^5.16.5",
  "@testing-library/react": "^13.4.0",
  "@testing-library/user-event": "^13.5.0",
  "axios": "^1.3.3",
  "react": "^18.2.0",
  "react-dom": "^18.2.0",
  "react-router-dom": "^6.8.1",
  "react-scripts": "5.0.1",
  "web-vitals": "^2.1.4"
},
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
"eslintConfig": {
  "extends": [
    "react-app",
    "react-app/jest"
  ]
},
"browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

## Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />


    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <script src="https://smtpjs.com/v3/smtp.js"></script>
  </body>
</html>
```

# BACKEND

## SERVER

### db.js

```
import mongoose from "mongoose";

import dotenv from 'dotenv';

dotenv.config();

const Connection = () => {

    const DB_URI =
'mongodb+srv://AyushMaurya:Ayush07@gmail.jrwcthr.mongodb.net/?retryWrites=true&
w=majority'

    try {

        mongoose.connect(DB_URI, { useNewUrlParser: true });

        mongoose.set('strictQuery', false);

        console.log('Database connected sucessfully');

    } catch (error) {

        console.log('Error while connecting with the database ', error.message)

    }

}

export default Connection;
```

## email-controller.js

```
import Email from "../model/email.js";

export const saveSendEmails = async (request, response) => {

    try {

        const email = await new Email(request.body);

        email.save();

        response.status(200).json('email saved successfully');

    } catch (error) {

        response.status(500).json(error.message);

    }

}

export const getEmails = async (request, response) => {

    try {

        let emails;

        if (request.params.type === 'starred') {

            emails = await Email.find({ starred: true, bin: false });

        } else if (request.params.type === 'bin') {

            emails = await Email.find({ bin: true })

        } else if (request.params.type === 'allmail') {

            emails = await Email.find({});

        } else if (request.params.type === 'inbox') {
            emails = [];
        } else {
            emails = await Email.find({ type: request.params.type });
        }
        response.status(200).json(emails);
    } catch (error) {
        response.status(500).json(error.message);
    }
}
export const toggleStarredEmail = async (request, response) => {
    try {
        await Email.updateOne({ _id: request.body.id }, { $set: { starred: request.body.value
}})
        response.status(201).json('Value is updated');
    } catch (error) {
        response.status(500).json(error.message);
    }
}
export const deleteEmails = async (request, response) => {
    try {
        await Email.deleteMany({ _id: { $in: request.body }})
        response.status(200).json('emails deleted successfully');
    } catch (error) {
        response.status(500).json(error.message);
    }
}
export const moveEmailsToBin = async (request, response) => {
    try {
        await Email.updateMany({ _id: { $in: request.body }}, { $set: { bin: true, starred: 
false, type: '' }});
    } catch (error) {
        response.status(500).json(error.message);
    }
}
```

## email.js

```
import mongoose from 'mongoose';
const EmailSchema = mongoose.Schema({
    to: {
        type: String,
        required: true
    },
    from: {
        type: String,
        required: true

    },

    subject: String,

    body: String,

    date: {

        type: Date,

        required: true

    },

    image: String,

    name: {

        type: String,

        required: true

    },

    starred: {

        type: Boolean,

        required: true,

        default: false

    },

    bin: {

        type: Boolean,

        required: true,
        default: false
    },
    type: {
        type: String,
        required: true,
    }
})
const email = mongoose.model('emails', EmailSchema);
export default email;
```

## routes.js

```
import express from 'express';
import { saveSendEmails, getEmails, toggleStarredEmail, deleteEmails,
    moveEmailsToBin } from '../controller/email-controller.js';
const routes = express.Router();
routes.post('/save', saveSendEmails);
routes.post('/save-draft', saveSendEmails);
routes.get('/emails/:type', getEmails);
routes.post('/starred', toggleStarredEmail);
routes.delete('/delete', deleteEmails);
routes.post('/bin', moveEmailsToBin);
```

```
export default routes;
```

## index.js

```
import express from 'express';

import cors from 'cors';

import Connection from './database/db.js';

import routes from './routes/route.js';

const app = express();

app.use(cors());

app.use(express.urlencoded());

app.use(express.json());

app.use('/', routes);

const PORT = 8000

Connection();

app.listen(PORT, () => console.log(`Server started on PORT ${PORT}`));
```

# CHAPTER 8

# TESTING

## 8.1 Unit Testing

- Individual units such as user authentication, email composition, and contact management were tested to ensure each function worked as intended.
- Tests covered scenarios like successful logins, proper email composition, and accurate contact additions.

## 8.2 Integration Testing:

- Different units were integrated to confirm they functioned together seamlessly.
- Tests included checking email composition, sending, and receiving to ensure end-to-end functionality.

## 8.3 Functional Testing:

- Key functionalities such as email composition, sending, receiving, and label organization were systematically tested.
- Test cases covered various scenarios to verify the correct functioning of these features.

## 8.4 Security Testing:

- Security measures, including HTTPS encryption, were rigorously tested to ensure the protection of user data.
- Common web vulnerabilities like XSS and CSRF were specifically addressed through testing.

## 8.5 User Interface Testing:

- The application's user interface was tested for responsiveness and compatibility across different devices and browsers.
- Tests ensured a consistent and visually appealing user experience.

## 8.6 Performance Testing:

- Performance tests were conducted to assess the application's responsiveness under varying loads.
- Scalability was verified to ensure the system could handle a growing user base.

## 8.7 User Acceptance Testing (UAT):

- UAT involved real users evaluating the application to ensure it met their expectations.
- Feedback from users was collected and used for further improvements.

## 8.8 Regression Testing:

- After implementing changes or additions, regression tests were executed to ensure existing functionalities were not adversely affected.

# CHAPTER 9

# CONCLUSION

In conclusion, a Gmail clone serves as a replicated version of Google's widely-used email service, Gmail. While these clones mimic Gmail's features and interface for various reasons, including providing alternatives or educational tools, it's important to note that they are not official Google products.

Users can engage with these clones to experience a similar email environment, with functionalities such as composing, sending, and organizing emails, all within a user interface reminiscent of Gmail. However, it's crucial to be mindful of the fact that these clones may not offer the same level of security, reliability, or official support as the original Gmail service provided by Google.

## 9.1 FUTURE SCOPE

The future scope of a Gmail clone involves continuous innovation, prioritizing user experience, and adapting to emerging technologies. Enhancements may include advanced security measures, AI-driven features, and cross-platform compatibility. Customization options, collaboration tools, and productivity integrations could transform it into a comprehensive digital workspace. Global accessibility, community engagement, and adherence to ethical standards will be pivotal. Mobile applications, IoT integration, and sustainable business models may feature prominently. Embracing emerging technologies like blockchain and AR/VR, alongside strategic partnerships, ensures relevance in a dynamic tech landscape. The clone's evolution will be guided by user feedback, environmental considerations, and compliance with evolving regulations.

# CHAPTER 10

# REFRENCES

- https://www.javatpoint.com/jsp-architecture
- https://www.tutorialspoint.com/servlets-tutorial
- https://www.edureka.com/jsp/
- https://www.researchgate.net/publication/
- https://www.geeksforgeeks.com/jsp-architecture/
- https://guru99.com/servlets/
- Developing dynamic website using an online website builder that is Weebly for Viking Fortune [Md. Shamsul Arafin Yi Jiang]- Laurea University of Applied Sciences
- Web Usability: A Literature Review [Giselle Joy Esmeria1 and Rosemary

R. Seva] Industrial Engineering Department, De La Salle University Science and Technology Complex