SPAM E-MAIL DETECTION USING MACHINE LEARING

A Project Report

Submitted in partial fulfillment of the
Requirements for the award of the Degree of

BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY)

By

ALANDI GANESH KADAM
Student Id: 20222317025
Roll No. 19013

Under the esteemed guidance of

Ms. Virgin Mary Fernando

DEPARTMENT OF INFORMATION TECHNOLOGY

S.I.W.S.

N.R. SWAMY COLLEGE OF COMMERCE AND ECONOMICS AND SMT. THIRUMALAI COLLEGE OF SCIENCE (AUTONOMOUS)

(Affiliated to University of Mumbai)

MUMBAI – 400031, MAHARASHTRA

NAAC RE-ACCREDITED A GRADE

2024-2025

S.I.W.S.

N. R. SWAMY COLLEGE OF COMMERCE AND ECONOMICS

AND

SMT. THIRUMALAI COLLEGE OF SCIENCE

(AUTONOMOUS)

(Affiliated to University of Mumbai)

377, Sewree -Wadala Estate, Mumbai Maharashtra - 400 031

NAAC Re-Accredited : A Grade

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This Is to certify that the project entitled, " **Spam E-mail Detection Using Machine Learning**", Is bonafied work of **ALANDI GANESH KADAM** bearing **Student Id : 20222317025** , **Seat. No.: 19013** submitted in partial fulfilment of the requirements for the award of degree of BACHELOR OF SCIENCE in INFORMATION TECHNOLOGY from University of Mumbai.

Internal Guide **Coordinator**

External Examiner

Date: ____________________ **College Seal**

ABSTRACT

- Spam emails, or unwanted messages, are a big problem for email users, so it's important to find good ways to automatically detect and filter them. In this project, i used machine learning techniques to identify spam emails. I use Naive Bayes algorithm.
- I began by preparing our email data, which involved cleaning and converting the text into a format that the algorithms could understand. I used method Term Frequency-Inverse Document Frequency (TF-IDF). Then, I compared how Ill each machine learning model performed in classifying mails as spam or not spam.
- results should that while Naive Bayes was simple and effective. The improved text processing techniques also helped the models work better.
- In summary, this project shows that machine learning can be a powerful tool for filtering out spam emails, making email systems more secure and easier to use.
- The goal of my project is to create a machine learning model that can accurately tell if an email is spam or not. This will help make email systems more efficient and secure.

Keywords : Spam Email Detection , Machine Learning, Naive Bayes, Text Processing, Term Frequency-Inverse Document Frequency (TF-IDF), Classification, Email Filtering, Model Accuracy, Text Cleaning, Automatic Email Filtering

ACKNOWLEDGEMENT

It’s my great pleasure to take this opportunity and sincerely thanks all those, who have showed me the way to successful project and helped me a lot during the completion of my project.I greatly thank my Project Guide **Ms.virgin fernando** without whom the completion of this project couldn’t have been possible.I take this opportunity to express my deep gratitude towards all the members of the Information Tech. Department, for helping me in the completion of the project. My sincere thanks to respect Principal **Dr. Manali Londhe** and Head of Information Tech. Department **Mrs. Santha Rani Mario** for providing all the facilities including availability of Computer Lab.I am greatly thanks teaching & Non-teaching staff of **Information Technology dept. of N. R. Swamy College of Commerce & Economics And Thirumalai College of Science**. Finally, I am thanks to my all Friends for their encouragement & support throughout the period of completion.

Thank you all for making this accomplishment possible.
Alandi Kadam

DECLARATION

I here by declare that the project entitled **, " Spam E-mail Detection Using Machine Learning"** done at DEPARTMENT OF INFORMATION TECHNOLOGY AT N.R.SWAMY TIRUMALAI COLLEGE OF SCIENCE (AUTONOMOUS) , has not been in any case duplicated to submit to any other university for the award of any degree . To the best oF my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfillment of the requirements for the award of degree of **BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY)** to be submitted as final semester project as part of our curriculum.

Name and Signature of the Student

TABLE OF CONTENTS

List Of Figures

Chapter 1: Introduction

- Spam emails, also known as junk mail, are unwanted and often annoying messages that fill up our email inboxes. These emails can be not only bothersome but also dangerous, as they sometimes contain harmful links or try to trick people into giving away personal information. As more spam emails are sent out, it's important to have a good way to automatically detect and filter them out.
- In this project,I am using machine learning to build a system that can automatically identify spam emails. Machine learning is a type of technology that helps computers learn from data and make decisions. By using machine learning, I aim to create a tool that can sort out spam from regular, important emails.
- I tested different machine learning techniques to see which one works best for detecting spam. Specifically, I used Naive Baye.I started by preparing our email data, which involved cleaning up the text and converting it into a format that the machine learning algorithms can understand. i used techniques like Term Frequency-Inverse Document Frequency (TF-IDF) to help with this.
- The goal of my project is to create a machine learning model that can accurately tell if an email is spam or not. This will help make email systems more efficient and secure. By the end of the project, I hope to show which machine learning method works best for spam detection and provide useful insights for managing emails better

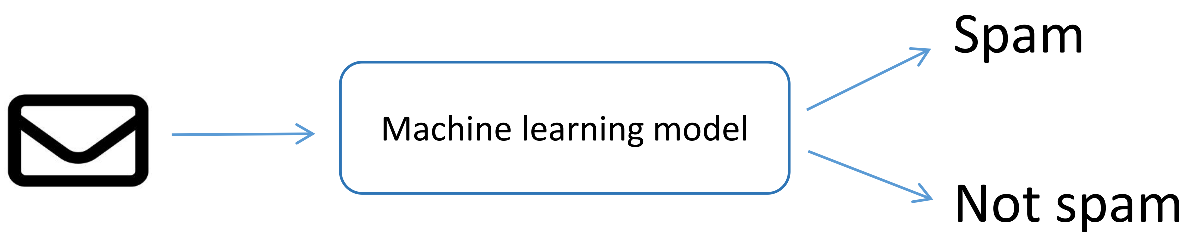


Figure 1.1

1.1 Background

- Spam mail detection has evolved significantly since the inception of email. In the early 1990s, as email usage began to rise, spammers started exploiting this communication medium to send unsolicited advertisements and malicious content. The sheer volume of spam emails became overwhelming for users, leading to a pressing need for effective spam detection systems.
- Initially, spam detection relied on **rule-based filters**, which identified spam based on specific keywords and phrases, such as "free," "limited time," or "guaranteed." While these early systems were a step forward, they had significant limitations. Spammers quickly learned to circumvent these filters by altering their language, making it easy to evade detection. This cat-and-mouse game highlighted the need for more sophisticated techniques.
- As technology advanced in the 2000s, machine learning began to play a crucial role in spam detection. Algorithms like **Naive Bayes** became popular for their ability to analyze large datasets and identify patterns that distinguish spam from legitimate emails (often referred to as "ham"). These machine learning models learn from the features of both spam and non-spam emails, allowing them to adapt and improve their accuracy over time. This marked a shift from simple keyword matching to a more nuanced approach that considers multiple factors.
- In addition to machine learning, spam detection systems began incorporating **blacklisting** techniques, where known spammers' IP addresses and domains were blocked. However, spammers often employed tactics like rotating IP addresses to evade these blacklists. As a result, spam detection systems also adopted **heuristic analysis**, which examines email characteristics and behaviors to identify potential spam, such as excessive links, suspicious attachments, or unusual sending patterns.
- Today, major email services like Gmail and Outlook utilize a combination of machine learning, natural language processing (NLP), and sender reputation systems to enhance spam detection. These services continuously analyze incoming emails in real-time, adjusting their filters based on user feedback and evolving spam tactics. Despite these advancements, spam detection remains a challenging field. Spammers continually develop new strategies to bypass filters, leading to an ongoing arms race between spam detection technologies and malicious actors.
- As a result, the development of effective spam detection systems is critical for maintaining the integrity of email communication and protecting users from potential threats. Researchers and developers continue to explore innovative approaches, such as deep learning and advanced feature extraction techniques, to stay one step ahead in the fight against spam. This ongoing evolution in spam detection not only enhances user experience but also contributes to overall cybersecurity efforts.

1.2 Objective

The primary objective of this project is to develop an intelligent email classification system that accurately identifies and filters spam messages using machine learning techniques. By analyzing email content, the system will classify messages as either spam or legitimate, enhancing user inbox security and reducing the risk of phishing, fraudulent, or irrelevant communications. This project aims to provide an efficient, automated solution that minimizes user reliance on manual spam filtering while improving detection accuracy and adaptability to evolving spam tactics.

1. Enhance Accuracy: Implement machine learning algorithm that can achieve high accuracy in distinguishing between spam and non-spam emails, minimizing false positives and false negatives.

2. Real-time Detection: Develop the system to detect and classify spam emails in real-time, ensuring prompt identification and filtering of unwanted emails as they arrive in a user's inbox.

3. Adaptability: Build a system that can continuously learn and adapt to new and evolving spam tactics by updating the model based on new datasets or user feedback.

4. User-friendly Interface: Create an intuitive and user-friendly application that allows users to easily interact with the system, such as typing in email content or automatically scanning inboxes for spam.

5. Efficient Data Processing: Optimize the system for fast processing of large volumes of email data, ensuring scalability and efficiency for both individual users and business applications.

6. Integration with Email Services: Develop the system to integrate seamlessly with common email services, providing users with a convenient and effective solution for spam filtering.

7. Security and Privacy: Ensure that user data, especially email content and metadata, is handled securely and that privacy is maintained throughout the spam detection process.

1.3 Purpose, Scope and Applicability

1.3.1 Purpose

The purpose of this project is to develop an effective machine learning model for spam email detection that enhances email security and user experience. As spam emails pose significant risks including phishing attacks, malware distribution, and cluttering of users' inboxes, an advanced and accurate detection system is crucial for mitigating these issues. This research aims to create a solution that not only improves the accuracy of spam detection but also reduces the incidence of false positives and false negatives, thereby providing a more reliable tool for managing email communications.

1.3.2 Scope

Dataset Collection and Preprocessing:

- Collect a dataset consisting of labeled emails (spam and legitimate).
- Preprocess the text data by removing stop words, punctuations, and special characters.
- Convert the preprocessed text into numerical features using TF-IDF, which will calculate the importance of words in the emails relative to the entire dataset.

Feature Extraction using TF-IDF:

- Apply TF-IDF to transform the raw email content into numerical vectors that represent the frequency and significance of words across emails.
- This feature extraction method will capture important terms in each email, helping the model distinguish between spam and legitimate content.

Model Training with Naive Bayes:

- Train the Naive Bayes algorithm on the TF-IDF feature set. Naive Bayes is a popular choice for text classification tasks like spam detection due to its simplicity and effectiveness with word frequency data.
- Implement and train the Multinomial Naive Bayes model, which is well-suited for this problem, as it works effectively with discrete features like word counts or frequencies from TF-IDF.

Evaluation and Model Performance:

- Evaluate the trained Naive Bayes model using test data to ensure it can accurately classify unseen emails as spam or legitimate.
- Key performance metrics such as accuracy, precision, recall, and F1-score will be used to assess the model's performance.
- Perform cross-validation to reduce overfitting and generalize the model to different datasets.

Real-time Spam Detection:

- Develop the system to classify incoming emails in real-time, using the pre-trained Naive Bayes model to predict whether an email is spam based on its TF-IDF features.
- Automatically move identified spam emails to a designated folder (e.g., "Spam" or "Trash").

User Interface Development:

- Create a user-friendly interface where users can either input an email message to classify or link their email accounts for automated spam detection.

- Display the classification results (spam or legitimate) and allow users to review and adjust the system if necessary.

API Integration:
- Allow integration of your API with various email platforms (e.g., Gmail, Outlook) using OAuth2 or API keys for authentication, enabling the system to scan users' inboxes and classify emails in real-time.
- API users will be able to send individual emails or bulk emails for spam classification, receiving a response with the predicted category.

Model Adaptation and Feedback:
- Implement a feedback loop where users can report false positives (legitimate emails marked as spam) and false negatives (spam emails marked as legitimate). This feedback will allow for retraining and continuous improvement of the model.
- Periodically retrain the Naive Bayes model with updated datasets to adapt to new spam tactics and patterns.

Security and Privacy:
- Ensure that users' email data is processed securely, with encryption and secure access controls to protect sensitive information.
- The system will comply with privacy regulations, ensuring that users' emails and credentials are not misused.

Deployment:
- The final system will be deployed as mobile application, allowing users to access spam detection features without needing to install local software.

1.3.3 Applicability

1. Personal Email Management: Individuals can use the spam detection tool to organize their inboxes by automatically moving spam messages to a "Trash" folder, improving inbox cleanliness and reducing manual filtering effort.

2. Customer Service Platforms: Businesses that handle large volumes of email inquiries can use the system to ensure that important customer messages are not mistakenly filtered into the spam folder while eliminating irrelevant or spammy emails.

3. Mobile Email Applications: Integrating the system into mobile email apps can enhance user experience by preventing spam and harmful content from reaching users' mobile devices.

4. Academic and Research Institutions: Educational institutions can use this system to manage and filter institutional emails, ensuring only relevant content reaches students, faculty, and administration, and preventing phishing attacks targeted at these groups.

1.3.3 Limitations

This Project has certain limitations.

i. This can only predict and classify spam but not block it.

ii. Analysis can be tricky for some alphanumeric messages and it may struggle with entity detection.

iii. Since the data is reasonably large it may take a few seconds to classify and anlayse the message.

iv. It doesn’t classiffies emails that contains images ,audio and videos.

v. Spammers continuously evolve their techniques to bypass filters. The model may struggle to detect novel or sophisticated spam strategies that were not present in the training data.

vi. The performance of the API may be affected by server load, response times, and network issues, which can hinder real-time email classification efficiency.

Chapter 2 : System Analysis

2.1 Exixting System

- The existing systems for spam email detection primarily rely on a combination of rule-based filtering, machine learning algorithms, and heuristic techniques, which are commonly used by popular email services like Gmail and Outlook. Early spam detection systems utilized rule-based filters that flagged emails based on specific keywords or phrases commonly found in spam, such as “free” or “urgent.”
- However, spammers quickly adapted by changing their language to bypass these filters, making this method less effective over time. Blacklisting is another method used in existing systems, where known spammer IP addresses and domains are blocked. While this helps prevent some spam, spammers often use rotating IP addresses to evade blacklists. Machine learning algorithms, such as Naive Bayes and Support Vector Machines (SVM), have also been integrated into modern spam detection systems, allowing these systems to learn patterns from large datasets of emails. Although machine learning enhances accuracy, it still faces challenges in identifying newer spam tactics.
- Heuristic analysis further assists spam detection by examining the structure and content of emails for suspicious patterns, but it can result in false positives, flagging legitimate emails as spam. Additionally, user feedback mechanisms are often employed to help refine spam filters, though this relies on manual input.

2.2 Proposed System

- In my proposed spam email detection system, I plan to use advanced machine learning techniques to effectively identify and classify emails as spam or ham. I'll focus on feature engineering to extract important characteristics from the emails, such as how often certain keywords appear, whether there are links or attachments, and the reputation of the sender. By looking at statistical features like the length of the email and the number of capitalized words, my model will learn to spot patterns that are typical of spam messages.
- I’ll use algorithms like Naive Bayes, Support Vector Machines (SVM), and Decision Trees to train my model on a diverse set of labeled emails, which will help it accurately classify incoming messages. To make sure my model performs well, I’ll use techniques like cross-validation and check its accuracy, precision, and recall. Plus, I want to add a feedback feature so that users can report any emails that are misclassified, which will help me gather data to continuously improve the model's accuracy over time. By taking this machine learning approach, I aim to create a scalable and efficient solution for spam detection, ultimately enhancing user experience and email security.

2.3 Requirements Analysis

2.3.1 Functional Requirement

1. User Authentication:The system must allow users to create an account and log in securely.Users should be able to reset their passwords if forgotten.

2. Email Submission:Users must be able to submit individual emails for spam detection.The system should accept plain text input or email file uploads.

3. Spam Classification:The system must classify submitted emails as either spam or legitimate.The classification result should include a confidence score (probability) indicating the likelihood of the email being spam.

4. Display Results:Upon classification, the system should display the result (spam or legitimate) to the user.The system should provide feedback about the classification process, including the features that contributed to the decision (if feasible).

5. Historical Data Storage:The system must store submitted emails and their classification results in a database for future reference.Users should be able to view their past submissions and classification results.

6. Spam Reporting:Users must be able to report false positives (legitimate emails classified as spam) and false negatives (spam emails classified as legitimate).The system should collect feedback on misclassifications to improve the model.

7. Admin Dashboard:Administrators must have access to a dashboard for monitoring system performance. The dashboard should provide insights into spam detection accuracy, user activity, and the ability to retrain the model with new data.

8. Model Retraining:The system should support periodic retraining of the spam detection model using new data collected from user submissions and feedback.
Administrators must be able to trigger retraining manually or set up a schedule.

9. API Functionality:The system must expose a RESTful API for external applications to interact with the spam detection service.The API should include endpoints for user authentication, email submission, and retrieving classification results.

10. User Notifications:Users should receive notifications for important actions (e.g., successful email submission, classification result, updates about their reported issues).

2.3.2 Non-Functional Requirement

1. Performance:The system should classify emails and return results within a specific time frame (e.g., within 2 seconds for typical email submissions) to ensure a responsive user experience.The system should handle a high volume of simultaneous users and submissions without degradation in performance.

2. Scalability:The architecture must support scalability to accommodate an increasing number of users and email submissions, ensuring that performance remains consistent as load increases.

3. Reliability:The system should ensure high availability, with a target uptime of at least 99.5% to minimize downtime and interruptions for users.It should provide accurate classifications consistently, maintaining a specified accuracy rate (e.g., over 90%) for spam detection.

4. Security:The system must implement strong security measures to protect user data, including encryption of sensitive information and secure authentication processes.

5. Usability:The user interface must be intuitive, user-friendly, and accessible, allowing users to navigate and use the system without extensive training or support.

6. Interoperability:The system should support integration with other applications and services through well-defined APIs, enabling seamless communication and data exchange.

7. Data Integrity:The system must ensure data integrity, preventing data loss or corruption during email submissions, storage, and processing.

2.4 Hardware Requirement

1. Web Server

The web server handles user requests, processes the application logic, and serves the front-end interface.

- Processor (CPU):Type: Multi-core processor (e.g., Intel Xeon or AMD Ryzen).Cores: Minimum of 4 cores.Clock Speed: At least 2.5 GHz.
- Memory (RAM):Capacity: Minimum of 16 GB (32 GB recommended for handling more concurrent users).
- Storage:Type: SSD (Solid State Drive) for fast read/write speeds.Capacity: Minimum of 256 GB (1 TB recommended for scaling and caching).
- Network Interface:Speed: Gigabit Ethernet (1 Gbps) or higher.Backup Power:UPS (Uninterruptible Power Supply) for ensuring uptime during power outages.

2. Database Server

The database server stores all user data, email records, and classification results.

- Processor (CPU):ype: Multi-core processor (e.g., Intel Xeon or AMD EPYC).Cores: Minimum of 6 cores.
- Clock Speed: At least 2.5 GHz.
- Memory (RAM):Capacity: Minimum of 32 GB (64 GB or more recommended for larger datasets and queries).
- Storage:Type: SSD for the database (potentially combined with HDD for backups).Capacity: Minimum of 512 GB (1 TB or more recommended based on data size).
- Network Interface:Speed: Gigabit Ethernet (1 Gbps) or higher for faster data access.
- Redundancy:RAID configuration (e.g., RAID 1 or RAID 10) for data redundancy.

3. Backup Server

The backup server is responsible for data backup and recovery processes.

- Processor (CPU):Type: Dual-core or quad-core processor.Cores: Minimum of 2 cores.Clock Speed: At least 2.0 GHz.
- Memory (RAM):Capacity: Minimum of 8 GB (16 GB recommended for faster processing of backups).
- Storage:Type: HDD or SSD, depending on budget and performance needs.Capacity: At least 1 TB (more depending on backup frequency and retention policies).

- Network Interface:Speed: Gigabit Ethernet (1 Gbps) or higher to facilitate quick data transfers.
- Backup Software:Ensure that the server can run backup software compatible with your database and web application.

2.5 Software Requirements

1. Operating System

Android:Minimum version: Android 8.0 (Oreo) or higher.

2. Development Environment

Integrated Development Environment (IDE):Android Studio: For developing Android applications.

3. Operating System (for Development)

Windows: Version 10 or higher (64-bit recommended).

4. Programming Languages

Backend:Java: For application logic and handling interaction between the app and Firebase.

Java Development Kit (JDK): Version 11 or later.

Frontend:XML: For designing the user interface (UI) of the mobile application.

5. Database

Firebase (Firebase Realtime Database or Firebase Firestore):Firebase SDK for Android: Integration with Firebase for data storage, user authentication, and cloud services.

Firebase Authentication: For user login and registration.

6. APIs

Firebase API:For database interactions, authentication, and real-time data syncing.

7. Libraries and Dependencies

AndroidX Libraries: Use the latest AndroidX libraries for backward compatibility and better performance.

Gson: For JSON parsing and serialization, useful if your app exchanges JSON data with Firebase.

OkHttp: For handling network operations

8. Firebase Configuration

Firebase Console:Setup the project in the Firebase Console.Downloading the google-services.json file and integrate it into Android Studio project.

9. Testing and Debugging Tools

JUnit: For unit testing in Java.

Espresso: For UI testing of Android application.

Firebase Test Lab: For testing app on a variety of virtual and physical devices.

10. Analytics and Monitoring
Firebase Analytics: For tracking user interactions, app usage, and detecting spam detection patterns.
Firebase Crashlytics: For monitoring and fixing crashes in your application.

11. Version Control
Git: For version control and managing source code.
GitHub or GitLab for hosting and collaboration.

12. Security
Firebase Security Rules: Set up Firebase Realtime Database or Firestore security rules to protect data.
SSL/TLS: Ensure secure data communication between the app and Firebase.

2.6 Justification of Selection of Technology

The choice of technologies for the Spam E-mail Detection application is critical to ensuring a robust, scalable, and user-friendly platform. This section provides a justification for the selected technologies based on their advantages, compatibility, and alignment with project goals.

1. Android Studio

- Android Studio is the official integrated development environment (IDE) for Android app development, which provides the most robust tools for building Android applications.
- It offers native support for Java (your backend language) and XML (your frontend design), ensuring seamless integration with your chosen technologies.
- Advanced debugging tools like Android Profiler and Emulator, which are critical for identifying issues early in development.
- Built-in Firebase integration: Android Studio offers easy setup and integration with Firebase, which simplifies your backend setup.

2. Java (Backend)

- Java is one of the most widely used programming languages for Android development. It is robust, well-documented, and has vast community support, ensuring that you'll find resources and help as needed.
- It has built-in libraries for networking, multithreading, and Firebase integration, which are essential for your app's spam detection, cloud database communication, and real-time updates.

- Strong backward compatibility: Java ensures that your app will run on older Android versions while still utilizing the latest Android APIs.
- High performance and scalability: Java can handle the computational tasks required for machine learning algorithms (such as Naive Bayes), ensuring efficient processing of spam detection.

3. XML (Frontend)

- XML is the standard markup language for designing user interfaces in Android development. It allows you to create structured, visually appealing layouts that are scalable across different screen sizes and resolutions.
- XML is easy to work with, especially in Android Studio, which provides a WYSIWYG (What You See Is What You Get) editor for easy drag-and-drop UI design and real-time previews.
- It enables clear separation between design (UI) and logic (Java code), promoting a clean architecture and maintainable codebase.

4. Firebase (Database & Backend Services)

- Real-time synchronization: Firebase Realtime Database or Firestore provides real-time updates, ensuring that data (such as detected spam messages) is always up to date across all user devices.
- Scalability: Firebase is a cloud-based solution that scales automatically with your application, eliminating the need for you to manage servers or handle infrastructure.
- Built-in authentication: Firebase Authentication simplifies user registration and login with support for email, phone number, Google, and other OAuth providers.
- Firebase Storage: Provides secure cloud storage for user data and detected spam messages, allowing easy access and sharing if needed.
- Ease of integration: Firebase SDKs integrate seamlessly into Android Studio, enabling quick setup without needing a custom backend. Its suite of services (Database, Authentication, Cloud Messaging, etc.) allows you to focus on core features rather than backend infrastructure.

5. Naive Bayes Algorithm

- Efficiency and simplicity: Naive Bayes is a probabilistic classifier that is both computationally efficient and easy to implement. It works well for text classification problems like spam detection, where it can learn patterns from the dataset and classify messages as spam or not spam.
- Suitability for small datasets: Even with a smaller dataset, Naive Bayes performs well, making it a great fit for your initial implementation and gradual scaling.
- Proven effectiveness: Naive Bayes has been widely used in email filtering, making it a reliable choice for your spam detection system.

6. TF-IDF (Term Frequency-Inverse Document Frequency)

- Effective feature extraction: TF-IDF is one of the most popular text feature extraction techniques. It converts text into numerical values (vectors) that can be used by the Naive Bayes algorithm to classify spam.
- Handling high-dimensional data: TF-IDF helps reduce noise in the data by focusing on the most important words, making it easier for the classifier to detect spam based on patterns in word frequency.
- Flexibility: TF-IDF works well with various machine learning algorithms, ensuring compatibility and adaptability as your project evolves.

Chapter 3 : System Design

3.1 Module Division

1. User Interface (UI) Module
Purpose:Handles the visual interaction between the user and the application. It allows users to authenticate, view and classify emails, check spam statistics, and compose/send emails securely.
Components:

- Login/Registration Screen:Users can log in or register using their email and password. Firebase Authentication is used to manage user credentials and sessions.
- Inbox Fragment:Displays all email messages fetched from the user's Gmail inbox (stored in Firebase). Emails are loaded in batches to ensure smooth performance. Non-spam messages are shown here.
- Trash Fragment (Spam Section):Shows only the emails classified as spam. These are stored under the "SpamEmails" node in Firebase after detection.
- Compose Email Screen:Allows users to compose and send emails. Before sending, the app checks the message content using the spam detection model. If the message is classified as spam, it prevents sending and displays a warning.
- Static Fragment (Statistics Dashboard):Displays analytics and statistics such as Total number of emails scanned ,Number of spam emails detected ,Spam detection rate, etc.
- Settings & Notifications:Users can manage notification preferences, such as alerts for new spam detections. They can also clear detection logs or stored messages if needed.

2. Authentication Module
Purpose: Handles user authentication, registration, and session management using Firebase.
Components:

- User Registration:Handles new user sign-ups by capturing email and password.Uses Firebase Authentication for registration.
- User Login:Allows existing users to log in.Validates credentials using Firebase.
- Session Management:Maintains active user sessions after login to avoid repetitive authentication.Logs out users after inactivity or when the user chooses to log out.

3. Spam Detection Module
Purpose: Performs the actual spam detection using the Naive Bayes algorithm with TF-IDF feature extraction.
Components:

- Preprocessing:Cleans and preprocesses the input message (e.g., removing stop words, lowercasing text).
- Extracts features using TF-IDF to transform the input text into numerical vectors.
- Spam Classification:Applies the Naive Bayes classifier to predict whether the message is spam or not based on pre-trained data.Returns the classification result (Spam or Not Spam).
- Model Training (optional):If needed, trains or updates the Naive Bayes model with new data in the background to improve accuracy.

4. Firebase Database Module

Purpose: Manages the storage and retrieval of user data, including past messages and detection results.
Components:

- Message Storage:Stores user-inputted messages along with the spam detection results in Firebase Realtime Database or Firestore.Each record includes message content, detection result, and timestamp.
- Spam Folder:Automatically stores detected spam messages in a designated "Spam" folder for user review.
- Data Retrieval:Retrieves the history of spam detections for each user from the database.

5. Notification Module

Purpose: Sends real-time notifications to the user when spam is detected or for general app alerts.
Components:

- Firebase Cloud Messaging (FCM):Sends push notifications to the user's mobile device when a message is detected as spam.
- In-App Notifications:Displays notifications within the app to alert users of detection results or other important actions.

6. API Integration Module

Purpose: Handles interactions between the frontend and backend components, such as connecting the mobile application to external services (Firebase, model APIs).
Components:

- HTTP Requests:Sends the user's message to the backend or local machine learning model for spam detection.Receives the result and displays it in the app.
- Firebase SDK:Connects the app to Firebase services such as Authentication, Realtime Database/Firestore, and Cloud Messaging.

7. Settings Module

Purpose: Provides user-configurable options for application behavior.

Components:

- Notification Preferences:Users can enable or disable spam detection notifications.
- Account Settings:Allows users to update their email, password, or log out of the app.

9. Backend Integration Module

Purpose: Integrates the mobile app with backend services, including Firebase and your spam detection API (if separate).

Components:

- Server Communication:If the app uses a dedicated server for more complex machine learning tasks, this module will handle API requests and responses to the server.
- Firebase Realtime Sync:Ensures real-time syncing between the mobile app and the Firebase backend for user authentication and spam detection data.

10. Security Module

Purpose: Ensures the security of user data and communication between the mobile app and Firebase.

Components:

- Firebase Security Rules:Implements security rules to ensure data access is restricted based on user authentication.
- Encryption:Ensures that sensitive data (e.g., user credentials) is encrypted before transmission to Firebase.

3.2 Data Dictionary

1. User Table

This table stores user information for authentication and profile management.

Field Name	Data Type	Description
UserID	Integer	Primary key, unique identifier for the user
UserName	String	Unique username for user login
Password	String	Encrypted password for the user
Email	String	User's email address for account verification
Phone	Integer	User's phone for user login
IsActive	BIT	Status indicating if the user account is active

2. Email Table

This table stores the messages submitted by users for spam detection and the results.

Field Name	Data Type	Description
Email_id	String	Unique identifier for each message (auto-generated).
user_id	String	Reference to the user who submitted the message.
message_text	Text	The content of the message submitted by the user.
Classification	Boolean	Ham or Spam

3. Spam Table

This table stores the spam messages that have been flagged and moved to a special folder for user review.

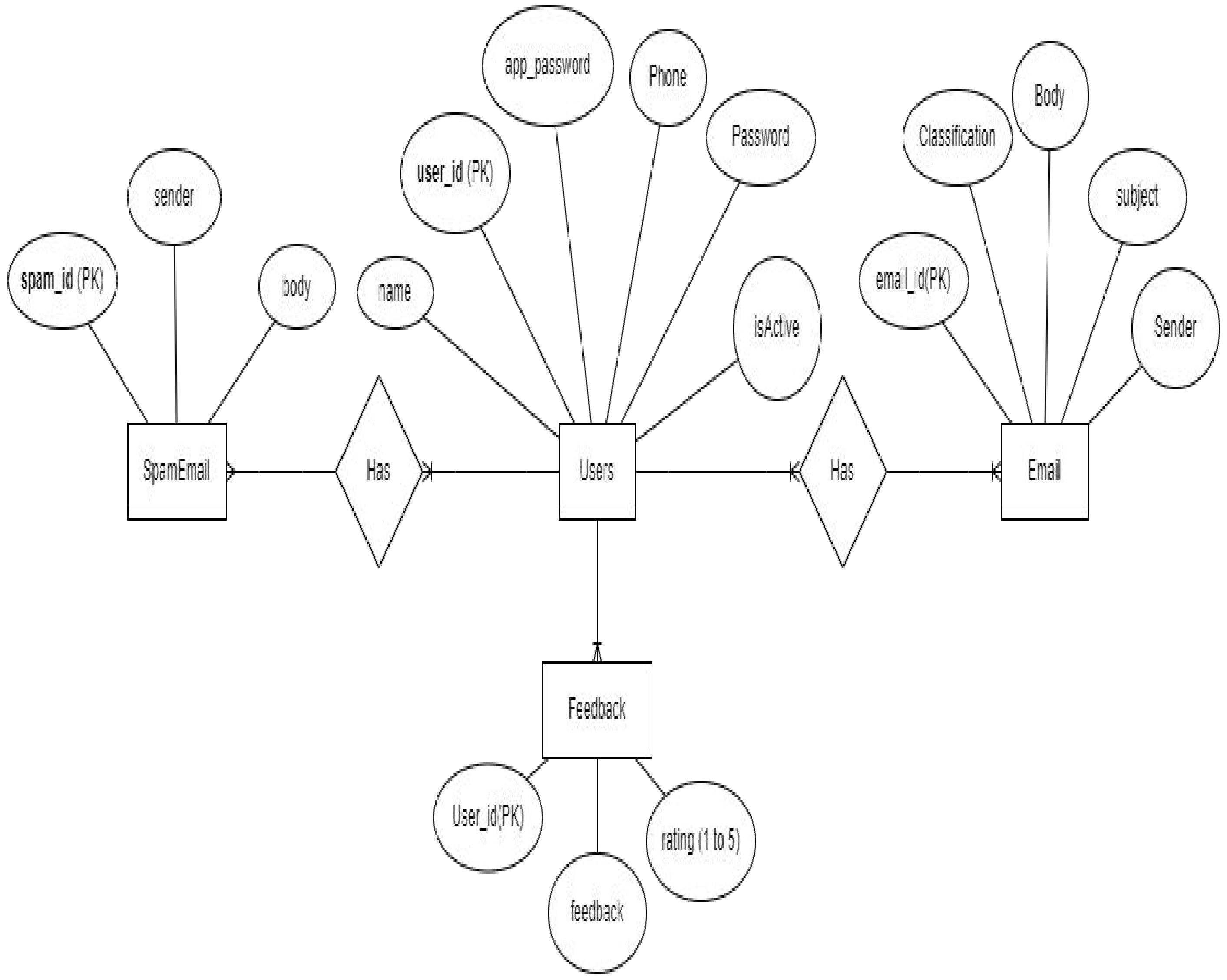
Field Name	Data Type	Description
Spam_id	String	Unique identifier for each message entry (auto-generated).
user_id	String	Reference to the user who owns the folder.
message_text	String	Reference to the spam message moved to the folder.

5. Notification Settings Table

This table stores user preferences for notification settings.

Field Name	Data Type	Description
setting_id	String	Unique identifier for each setting (auto-generated).
user_id	String	Reference to the user for whom the settings apply.
notifications_on	Boolean	Whether notifications are enabled (True) or disabled (False).
last_updated	Timestamp	Timestamp when the settings were last updated.

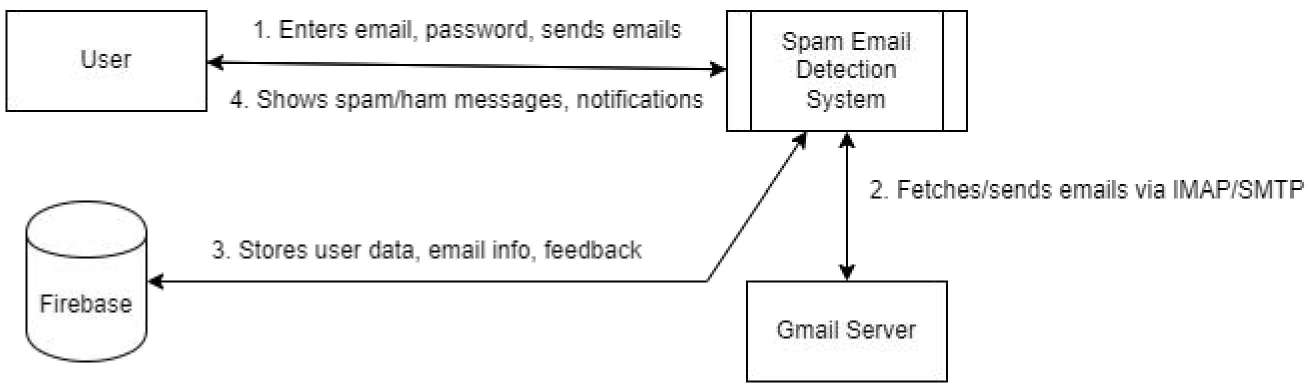
3.3 Entity Relationship (ER) Diagram



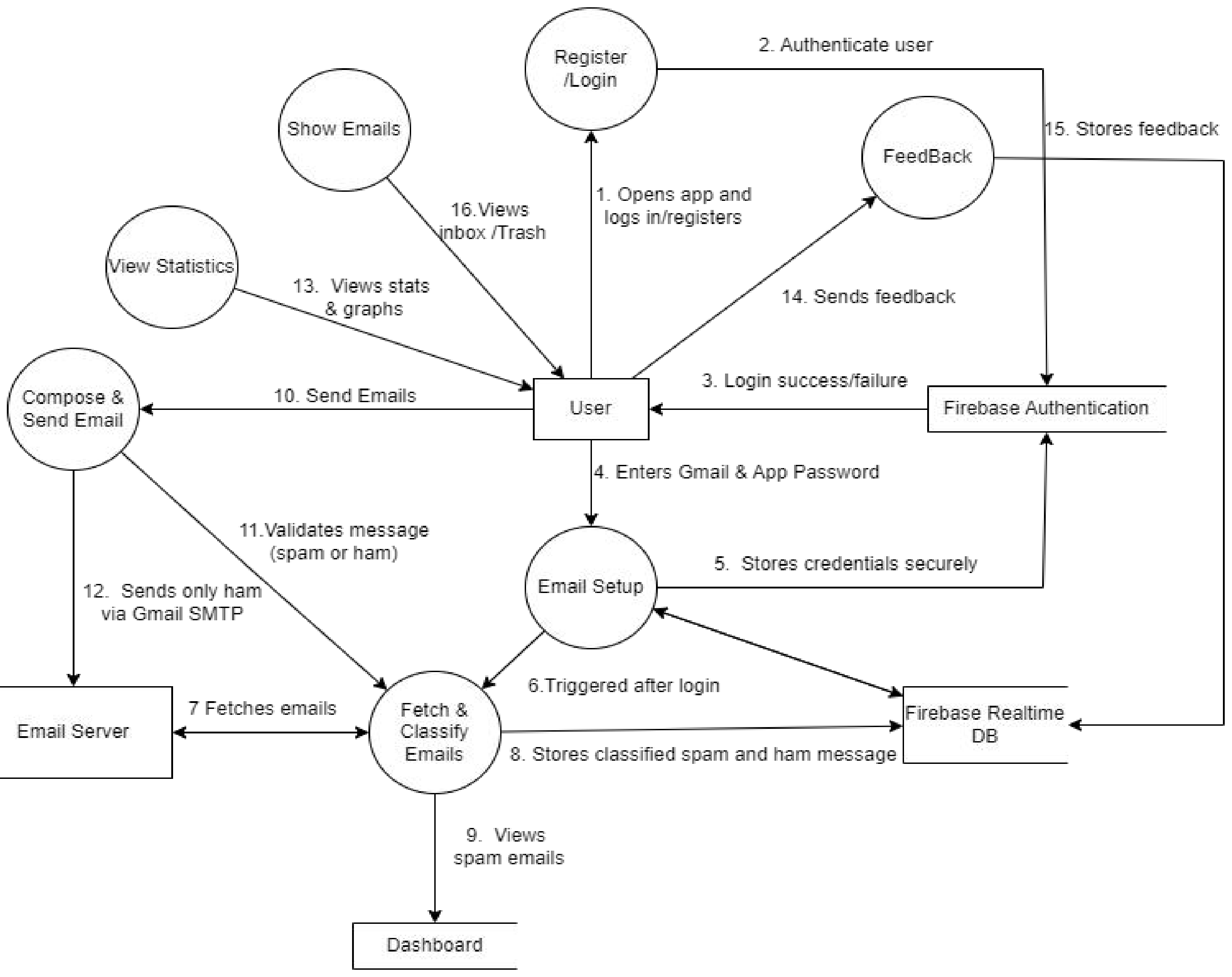
3.4 Data Flow Diagram/UML Diagram

Data Flow diagram

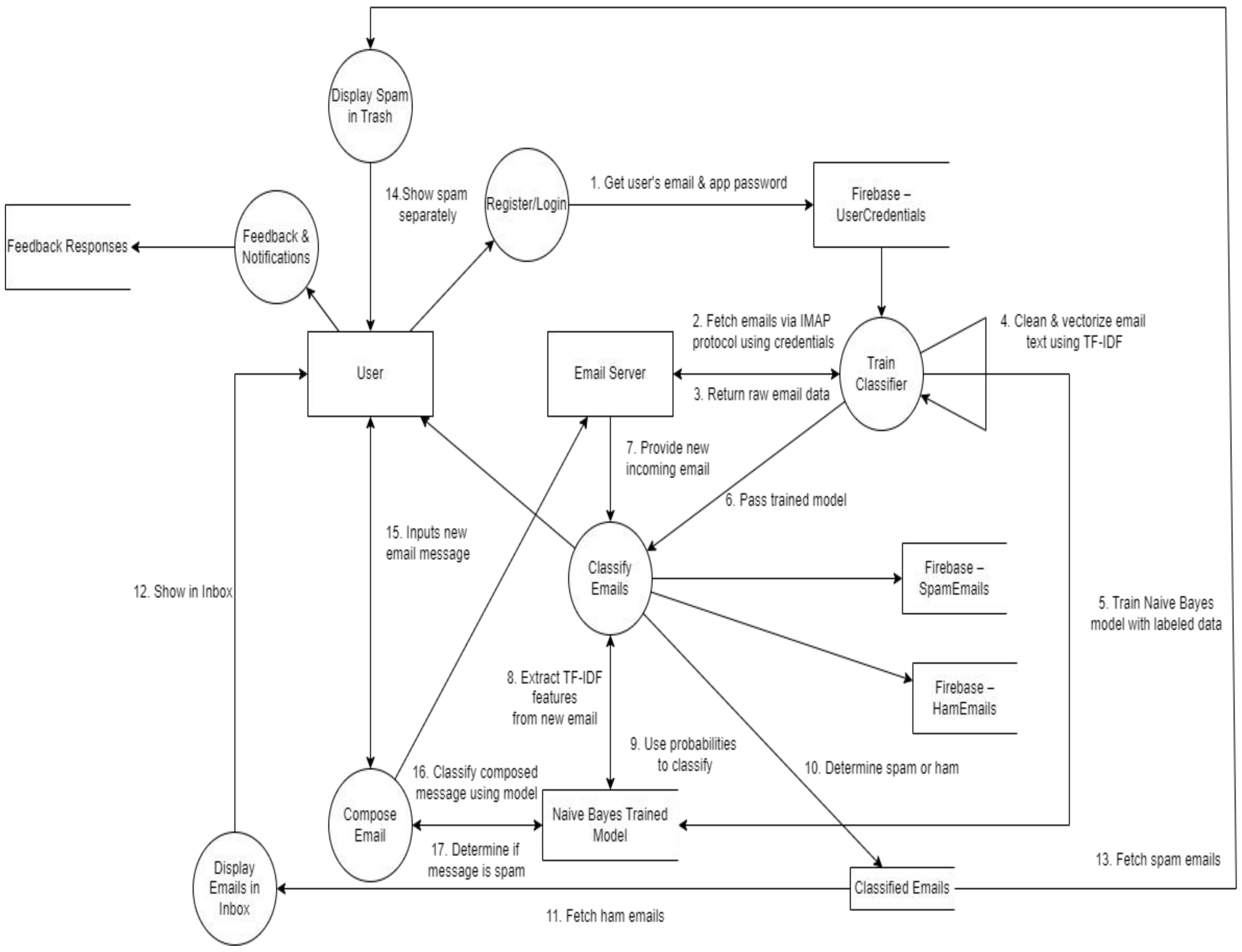
Data Flow Diagram Level 0


User
1. Enters email, password, sends emails
4. Shows spam/ham messages, notifications
Spam Email Detection System
2. Fetches/sends emails via IMAP/SMTP
3. Stores user data, email info, feedback
Firebase
Gmail Server

Data Flow Diagram Level 1



Data Flow Diagram Level 2


Display Spam in Trash
14.Show spam separately
Register/Login
1. Get user's email & app password
Firebase – UserCredentials
Feedback Responses
Feedback & Notifications
2. Fetch emails via IMAP protocol using credentials
4. Clean & vectorize email text using TF-IDF
User
Email Server
Train Classifier
3. Return raw email data
7. Provide new incoming email
6. Pass trained model
15. Inputs new email message
Classify Emails
Firebase – SpamEmails
12. Show in Inbox
5. Train Naive Bayes model with labeled data
Firebase – HamEmails
8. Extract TF-IDF features from new email
9. Use probabilities to classify
10. Determine spam or ham
16. Classify composed message using model
Compose Email
Naive Bayes Trained Model
17. Determine if message is spam
Display Emails in Inbox
Classified Emails
13. Fetch spam emails
11. Fetch ham emails

3.4.1 Class Diagram

Class diagrams are a type of UML (Unified Modeling Language) diagram used in software engineering to visually represent the structure and relationships of classes within a system i.e. used to construct and visualize object-oriented systems.

Class diagrams provide a high-level overview of a system's design, helping to communicate and document the structure of the software.

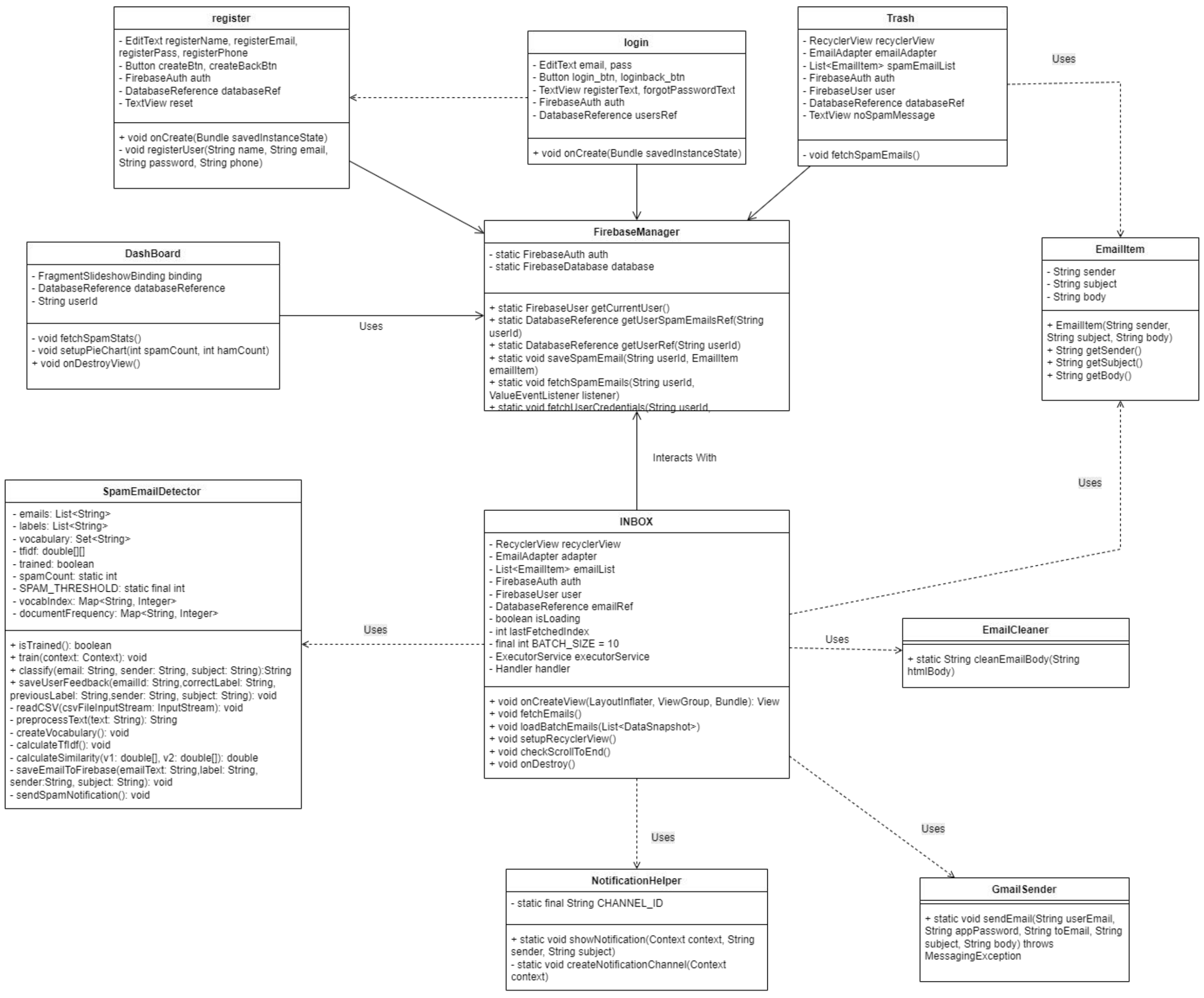

register
- EditText registerName, registerEmail, registerPass, registerPhone
- Button createBtn, createBackBtn
- FirebaseAuth auth
- DatabaseReference databaseRef
- TextView reset
+ void onCreate(Bundle savedInstanceState)
- void registerUser(String name, String email, String password, String phone)
login
- EditText email, pass
- Button login_btn, loginback_btn
- TextView registerText, forgotPasswordText
- FirebaseAuth auth
- DatabaseReference usersRef
+ void onCreate(Bundle savedInstanceState)
Trash
- RecyclerView recyclerView
- EmailAdapter emailAdapter
- List<EmailItem> spamEmailList
- FirebaseAuth auth
- FirebaseUser user
- DatabaseReference databaseRef
- TextView noSpamMessage
- void fetchSpamEmails()
Uses
DashBoard
- FragmentSlideshowBinding binding
- DatabaseReference databaseReference
- String userId
- void fetchSpamStats()
- void setupPieChart(int spamCount, int hamCount)
+ void onDestroyView()
Uses
FirebaseManager
- static FirebaseAuth auth
- static FirebaseDatabase database
+ static FirebaseUser getCurrentUser()
+ static DatabaseReference getUserSpamEmailsRef(String userId)
+ static DatabaseReference getUserRef(String userId)
+ static void saveSpamEmail(String userId, EmailItem emailItem)
+ static void fetchSpamEmails(String userId, ValueEventListener listener)
+ static void fetchUserCredentials(String userId,
EmailItem
- String sender
- String subject
- String body
+ EmailItem(String sender, String subject, String body)
+ String getSender()
+ String getSubject()
+ String getBody()
Interacts With
Uses
SpamEmailDetector
- emails: List<String>
- labels: List<String>
- vocabulary: Set<String>
- tfidf: double[][]
- trained: boolean
- spamCount: static int
- SPAM_THRESHOLD: static final int
- vocabIndex: Map<String, Integer>
- documentFrequency: Map<String, Integer>
+ isTrained(): boolean
+ train(context: Context): void
+ classify(email: String, sender: String, subject: String):String
+ saveUserFeedback(emailId: String,correctLabel: String, previousLabel: String,sender: String, subject: String): void
- readCSV(csvFileInputStream: InputStream): void
- preprocessText(text: String): String
- createVocabulary(): void
- calculateTfIdf(): void
- calculateSimilarity(v1: double[], v2: double[]): double
- saveEmailToFirebase(emailText: String,label: String, sender:String, subject: String): void
- sendSpamNotification(): void
Uses
INBOX
- RecyclerView recyclerView
- EmailAdapter adapter
- List<EmailItem> emailList
- FirebaseAuth auth
- FirebaseUser user
- DatabaseReference emailRef
- boolean isLoading
- int lastFetchedIndex
- final int BATCH_SIZE = 10
- ExecutorService executorService
- Handler handler
+ void onCreateView(LayoutInflater, ViewGroup, Bundle): View
+ void fetchEmails()
+ void loadBatchEmails(List<DataSnapshot>)
+ void setupRecyclerView()
+ void checkScrollToEnd()
+ void onDestroy()
Uses
EmailCleaner
+ static String cleanEmailBody(String htmlBody)
Uses
Uses
NotificationHelper
- static final String CHANNEL_ID
+ static void showNotification(Context context, String sender, String subject)
- static void createNotificationChannel(Context context)
GmailSender
+ static void sendEmail(String userEmail, String appPassword, String toEmail, String subject, String body) throws MessagingException

Fig 3.4.1

3.4.2 Use Case Diagram

A Use Case Diagram is a type of UML diagram that represents the interaction between actors (users or external systems) and a system under consideration to accomplish specific goals. It provides a high-level view of the system's functionality by illustrating the various ways users can interact with it.

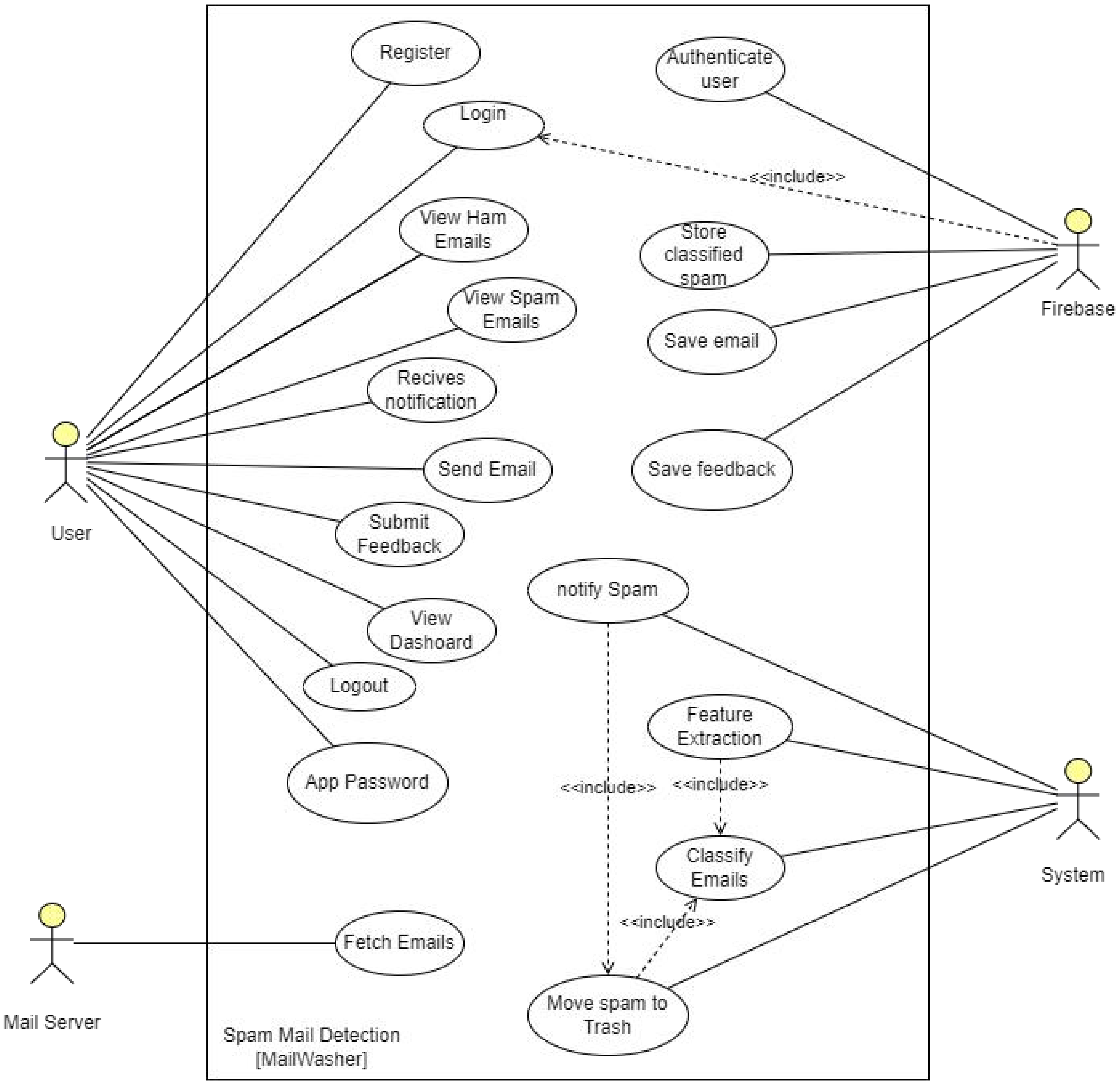


Fig : 3.4.2

3.4.3 Sequence Diagram

A sequence diagram is a most commonly used interaction diagram.

An interaction diagram is used to show the interactive behavior of a system. Since visualizing the interactions in a system can be difficult, we use different types of interaction diagrams to capture various features and aspects of interaction in a system.

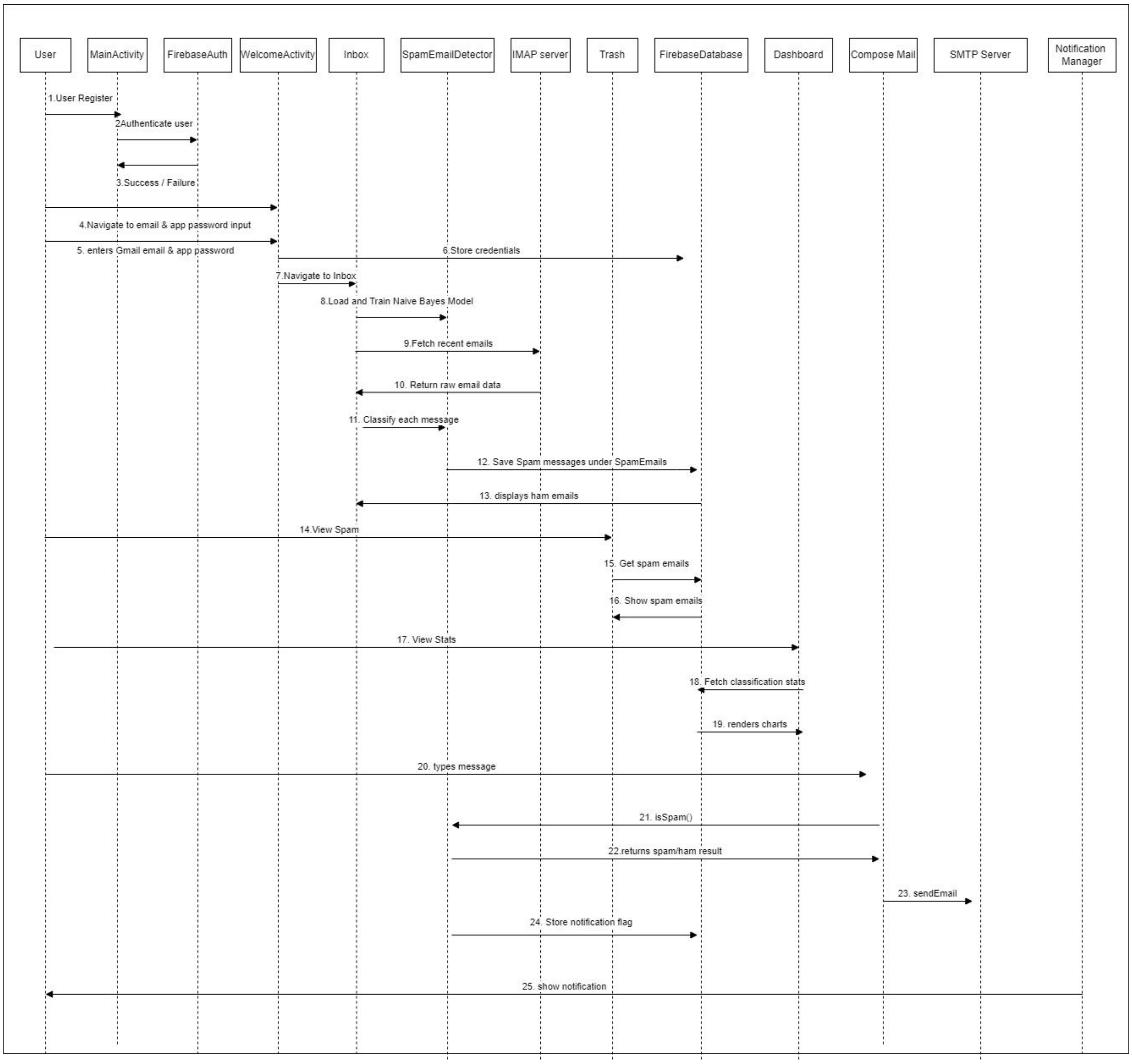


Fig : 3.4.3

3.4.4 State Transition Diagram

A **State Transition Diagram (STD)** is a graphical representation used in system design to illustrate the states of a system and the transitions between those states based on events or conditions. It provides a clear view of how a system behaves in response to external inputs, highlighting:

- **States**: The various conditions or statuses that an entity (like a system, object, or component) can be in at any given time.
- **Transitions**: The movement from one state to another, triggered by events or actions.
- **Events**: The occurrences that cause transitions between states.
- **Actions**: Activities that result from entering or leaving a state

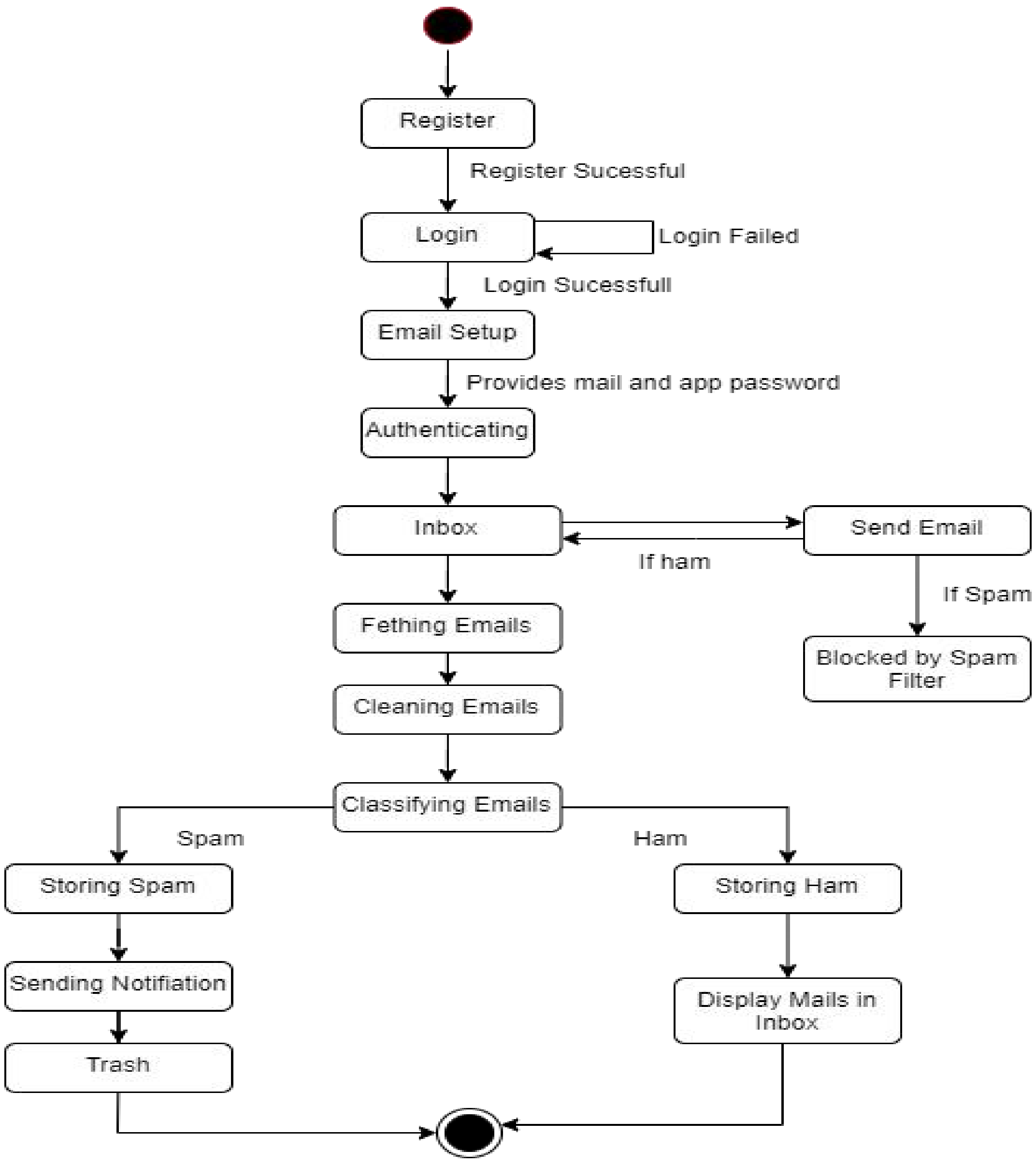


Fig : 3.4.4

3.4.5 Activity Diagram

An **Activity Diagram** is a type of UML (Unified Modeling Language) diagram that represents the dynamic aspects of a system by illustrating the flow of activities or actions.

- **Activities**: Represented as rounded rectangles, these denote the tasks or actions performed in the process.
- **Transitions**: Arrows that show the flow from one activity to another, indicating the order of execution.
- **Decision Nodes**: Diamonds that represent points where a choice must be made, leading to different branches of execution based on conditions.
- **Start and End Nodes**: Circles that signify the initiation and completion of the workflow.
- **Swimlanes**: Vertical or horizontal divisions that categorize activities based on the responsible actors or components.

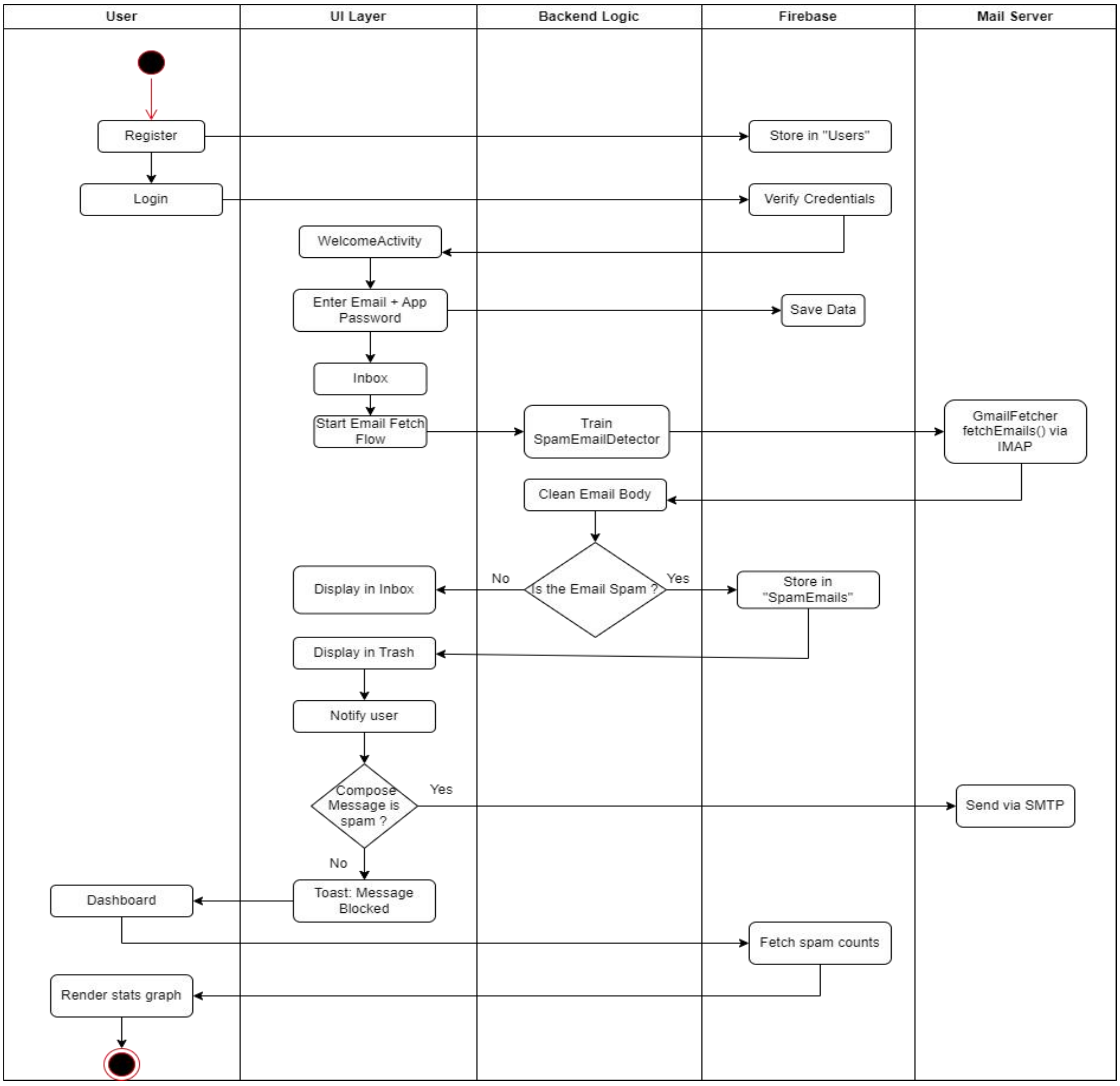


Fig : 3.4.5

3.4.6 Component Diagram

A **Component Diagram** is a type of UML (Unified Modeling Language) diagram that depicts the organization and dependencies among various components in a system. It focuses on the high-level structure of the system, illustrating how components interact and collaborate to fulfill system functionality

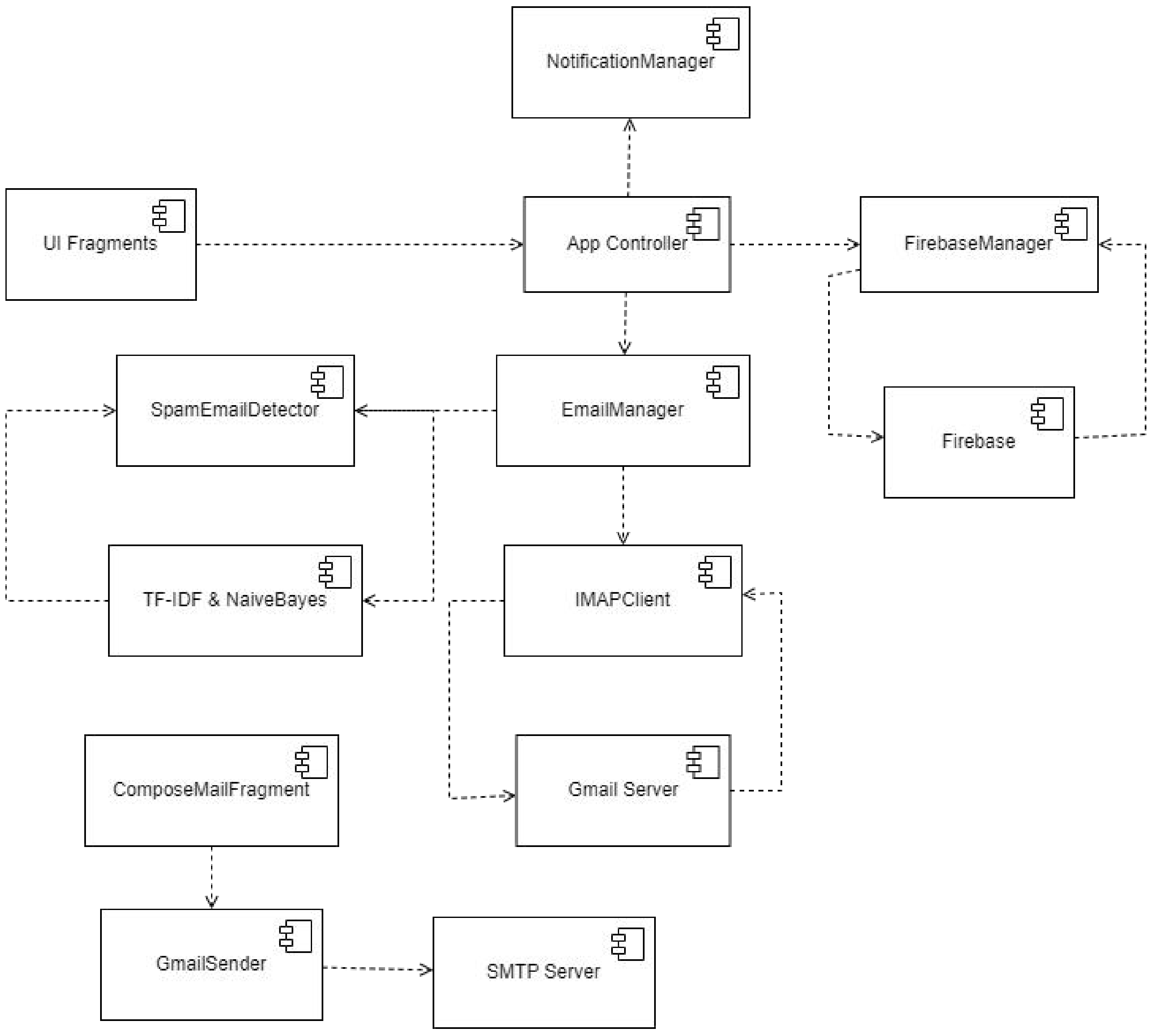


Fig : 3.4.6

3.4.7 Deployement Diagram

A **Deployment Diagram** is a type of UML (Unified Modeling Language) diagram that illustrates the physical deployment of artifacts on nodes in a system. It focuses on the hardware components, their relationships, and how software artifacts are distributed across them

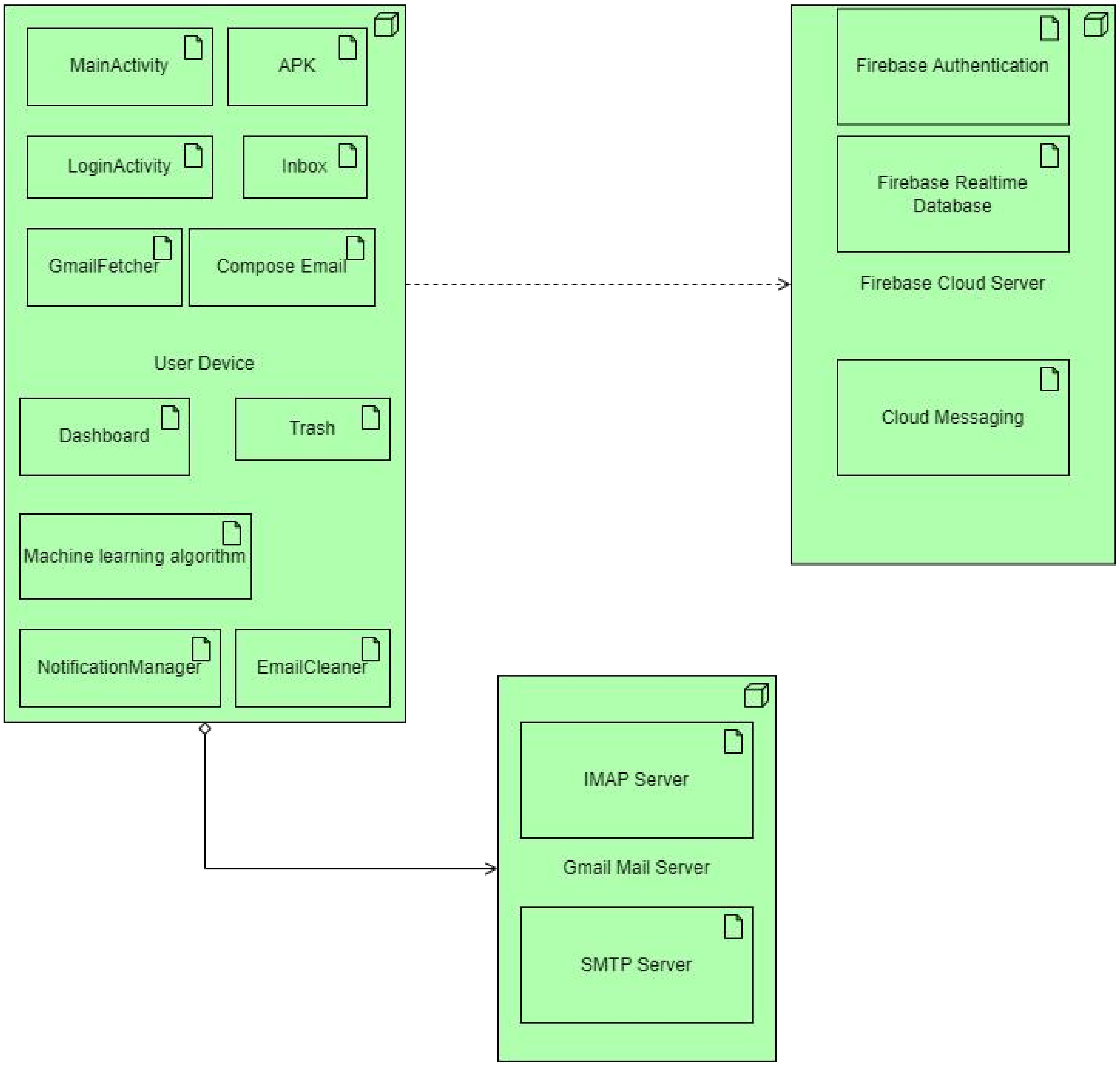


Fig : 3.4.7

3.4.8 CoCoMo Model

COCOMO has three models:

- Organic: For small, simple projects with a small team1.
- Semi-Detached: For medium projects with mixed complexity.
- Embedded: For complex projects that must operate within existing systems.

Spam E-mail detection is a moderately complex application. We chose the Organic model, which is appropriate for small to medium-sized projects.

For a organic project, the constants are: a = 2.4, b = 1.05, c = 2.5, d = 0.38

KLOC project consists of 1000 lines of code (KLOC = 1).

1.Effort Calculation : Effort=a×(KLOC) b Effort=2.4×(1) 1.05=2.4 person-months

2. Time to Develop (TDEV) : TDEV=c×(Effort)d TDEV=2.5×(3.0) 0.38=3.37 months

- Results:

Effort: Approximately 2.4 person-months.

Development Time (TDEV): Approximately 3.37 months.

3.4.9 Gantt Chart

Process	Quarter 1				Quarter 2				Quarter 3			
	Jan	Feb	Mar	April	May	June	July	Aug	Sep	Oct	Nov	Dec
Project Planning & Setup		■	■									
Data Collection				■	■							
Model Development						■	■	■				
Model Optimization									■	■		
Integration with System									■	■		
Testing										■	■	
Deployment											■	■
Documentation												■

Fig : 3.4.9

3.4.10 Decision Table

Conditions	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8	Rule 9	Rule 10	Rule 11	Rule 12
User Exists	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes
Valid Password	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No	No
Contains Spam Keywords	Yes	Yes	Yes	No	No	No	No	Yes	No	Yes	Yes	Yes
From Unknown Sender	Yes	No	No	Yes	Yes	No	Yes	No	Yes	No	Yes	Yes
Subject Length > 50	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	Yes
Contains Attachments	Yes	Yes	No	Yes	No	No	No	No	No	Yes	No	No
User Marked as Spam	Yes	No	No	Yes	No	No	No	Yes	No	No	Yes	Yes
Actions												
Allow Login	Yes	Yes	Yes	Yes	Yes	No	No	No	No	Yes	No	No
Register User	-	-	-	-	-	-	Yes	Yes	Yes	-	-	Yes
Show Home Page	Yes	Yes	Yes	Yes	Yes	No	No	No	No	Yes	No	No
Move Email to Trash Folder	Yes	Yes	Yes	Yes	No	-	-	Yes	No	Yes	Yes	Yes
Move Email to Trash	Yes	Yes	Yes	Yes	No	-	-	Yes	No	Yes	Yes	Yes
Notify User of Spam	Yes	Yes	Yes	Yes	No	-	-	Yes	No	Yes	Yes	Yes
Keep Email in Inbox	No	No	No	No	Yes	-	-	No	Yes	No	No	No

Fig : 3.4.10

Chapter 4 : Implementation and Testing

4.1 Code (Place Core segments)

This section outlines key parts of the application's source code, demonstrating how core functionalities such as user authentication, spam email detection, and email handling are implemented.

1. User Authentication (Login & Registration)
Firebase Authentication is used to manage user login and registration securely.

```
FirebaseAuth mAuth = FirebaseAuth.getInstance();

mAuth.createUserWithEmailAndPassword(email, password)

   .addOnCompleteListener(task -> {

       if (task.isSuccessful()) { // Registration successful}

 else { // Handle errors }

 });

mAuth.signInWithEmailAndPassword(email, password)

   .addOnCompleteListener(task -> {

       if (task.isSuccessful {  // Login successful }

else {  // Handle errors }

  });
```

2. Spam Detection Logic (Using TF-IDF & Naïve Bayes)

The following segment uses a machine learning model to classify emails based on content.

```
public boolean isSpam(String message) {

   Map<String, Double> tfidf = computeTFIDF(message);

   double spamScore = computeCosineSimilarity(tfidf, spamModel);

   double hamScore = computeCosineSimilarity(tfidf, hamModel);

   return spamScore > hamScore;

}
```

If the email is detected as spam, it is not sent or moved to the inbox.

3. Email Sending with Spam Check

Before an email is sent, its content is classified. If spam is detected, sending is blocked.

```
sendButton.setOnClickListener(v -> {

   String message = bodyEditText.getText().toString().trim();

   if (SpamEmailDetector.isSpam(message)) {

      Toast.makeText(this, "Sending blocked: The message looks like spam.",
      Toast.LENGTH_LONG).show();

 Return; }

   new Thread(() -> {

      try {

         GmailSender.sendEmail(senderEmail, appPassword, to, subject, message);

         runOnUiThread(() -> Toast.makeText(this, "Email sent!", Toast.LENGTH_SHORT).show());

      } catch (Exception e) {

         runOnUiThread(() -> Toast.makeText(this, "Error: " + e.getMessage(),
Toast.LENGTH_LONG).show());

      }

   }).start();

});
```

4. Email Fetching from Gmail using IMAP

The app uses IMAP protocol to read Gmail messages and classify them.

```
Store store = session.getStore("imaps");

store.connect("imap.gmail.com", userEmail, appPassword);

Folder inbox = store.getFolder("INBOX");

inbox.open(Folder.READ_ONLY);

Message[] messages = inbox.getMessages(startIndex, endIndex);

for (Message message : messages) {

   String body = EmailCleaner.cleanEmailBody(message);

   if (SpamEmailDetector.isSpam(body)) {
        // Store in Firebase SpamEmails }
 else {    // Show in Inbox  }
}
```

5. Firebase Storage of Spam Emails

Spam messages are saved to Firebase under the current user's ID for later reference and statistics.

```
DatabaseReference ref = FirebaseDatabase.getInstance().getReference("Users")
    .child(userId).child("SpamEmails");
ref.push().setValue(new EmailItem(sender, subject, body));
```

6. Fragment Navigation and Analysis Dashboard

Spam and ham message statistics are displayed in a dedicated fragment using simple data visualization.

```
TextView spamCount = view.findViewById(R.id.spamCount);
TextView hamCount = view.findViewById(R.id.hamCount);
// Values loaded from Firebase and updated dynamically
spamCount.setText(String.valueOf(spamTotal));
hamCount.setText(String.valueOf(hamTotal));
```

These core code segments illustrate the complete functionality of the spam email detection Android application. The code was written in a modular, scalable, and maintainable way to ensure robustness and ease of testing.

4.2 Testing Approach

The testing phase of the project was conducted at two primary levels: unit testing and integration testing.

1. Unit Testing

Individual components of the system were tested in isolation to verify their functionality. Key areas tested include:

1.1 User Authentication: Tested the login functionality with both valid and invalid credentials to ensure the system accurately verifies user inputs and provides appropriate feedback.

1.2 Spam Detection Logic: Emails with known spam characteristics were used to verify if the machine learning model correctly identifies and flags them as spam.

1.3 Edge Case Handling: Specific tests were conducted to evaluate the behavior of the system in less common scenarios such as:

- Emails with excessively long subject lines
- Emails from unknown or suspicious senders
- Messages with empty or HTML-rich bodies

2. Integration Testing:

After verifying individual components, integration testing ensured that all modules worked together seamlessly. For example:

- Spam emails were correctly detected and automatically moved to the trash folder.
- Legitimate (ham) emails remained in the inbox.
- Notifications were shown appropriately when spam was detected or blocked.

Overall, the testing process validated that the system behaves as expected under various conditions and handles both typical and edge-case inputs effectively.

4.2.1 Unit Testing

Unit Testing is a crucial part of the software development lifecycle, used to validate the functionality of individual modules or components in isolation. For this Spam Email Detection Android Application, unit testing was applied to verify the correctness of the following critical components:

1. **User Authentication:**

Unit tests were written to ensure that the login and registration features function properly with both valid and invalid credentials. This helped verify that user inputs are handled securely and that authentication flows are robust.

Code :

```
auth.signInWithEmailAndPassword(emailInput, passInput)
        .addOnSuccessListener(authResult -> {
            FirebaseUser user = auth.getCurrentUser();
            if (user != null)
            {   String userId = user.getUid();
                usersRef.child(userId).addListenerForSingleValueEvent(new ValueEventListener()
                    {  @Override
                        public void onDataChange(@NonNull DataSnapshot snapshot) {
                        if (snapshot.exists())
                        { Boolean isActive = snapshot.child("isActive").getValue(Boolean.class);
                            String username = snapshot.child("name").getValue(String.class);  // Fetch name
                            String userEmail = snapshot.child("email").getValue(String.class); // Fetch email
                            if (Boolean.TRUE.equals(isActive))
                        {   // Save user data in SharedPreferences
                           SharedPreferences prefs =getSharedPreferences("UserData",MODE_PRIVATE);
                           SharedPreferences.Editor editor = prefs.edit();
                           editor.putString("username", username);
                           editor.putString("email", userEmail);
                           editor.apply();
                           Toast.makeText(login.this, "Login Successful", Toast.LENGTH_SHORT).show();
                           // Pass data to DrawerActivity
                           Intent intent = new Intent(login.this, home.class);
                           intent.putExtra("username", username);
                           intent.putExtra("email", userEmail);
                           startActivity(intent);
                           finish();
                        }
                     else { Toast.makeText(login.this, "Your account is inactive. Contact support.",
                           Toast.LENGTH_LONG).show();
                           auth.signOut()   }
                  }
                 else{ Toast.makeText(login.this, "User record not found.",
Toast.LENGTH_LONG).show();
                       auth.signOut();   }
              }
                 @Override
                 public void onCancelled(@NonNull DatabaseError error)
                 {  Toast.makeText(login.this, "Failed to fetch user data: " + error.getMessage(),
.                          LENGTH_LONG).show();
                 }
               });
            }
         })
         .addOnFailureListener(e ->
               Toast.makeText(login.this, "Login Failed: " + e.getMessage(),
Toast.LENGTH_LONG).show()
         );
});
```

Test Case :

Test Case ID	Test Scenario	Test Steps	Expected Result
UA_TC_01	Register with valid details	Enter valid name, email, password, and phone. Click "Create Account".	User is registered and redirected to Login screen.
UA_TC_02	Missing fields during registration	Leave one or more fields empty. Click "Create Account".	Toast: "All fields are required!"
UA_TC_03	Invalid phone number format	Enter non-10-digit or invalid phone. Click "Create Account".	Toast: "Phone number must be exactly 10 digits!"
UA_TC_04	Register with already used email	Enter an already registered email. Click "Create Account".	Toast: "The email address is already in use..."
UA_TC_05	Login with correct credentials	Enter valid email and password. Click "Login".	User is logged in and redirected to Home screen.

UA_TC_06	Incorrect password	Enter valid email and wrong password. Click "Login".	Toast: "The password is invalid..."
UA_TC_07	Unregistered email login attempt	Enter an email not in Firebase. Click "Login".	Toast: "There is no user record..."
UA_TC_08	Login with empty fields	Leave email or password blank. Click "Login".	Toast: "Email and Password are required."
UA_TC_09	Inactive user account login	Login with an account having isActive = false.	Toast: "Your account is inactive. Contact support."
UA_TC_10	Forgot password - valid email	Enter registered email. Click "Reset Password".	Toast: "Reset link sent to your email."
UA_TC_11	Forgot password - empty email	Leave email field empty. Click "Reset Password".	Toast: "Enter your email to reset password."
UA_TC_12	Session persistence after login	Login and check SharedPreferences for email/username.	Username and email stored locally for session.

2. Spam Detection Algorithm:

The core spam classification logic was tested by simulating various input scenarios—emails with spam keywords, suspicious links, unknown senders, and long subjects. These tests verified that the model correctly identifies spam and ham messages with high accuracy.

```
public String classify(String email, String sender, String subject) {
    email = preprocessText(email);
    String[] words = email.split("\\s+");
    double[] emailTfidf = new double[vocabIndex.size()];
    for (String word : words) {
        Integer wordIndex = vocabIndex.get(word);
        if (wordIndex != null && wordIndex < emailTfidf.length) {
            emailTfidf[wordIndex] += 1;
        }
    }
    if (tfidf == null || tfidf.length == 0) {
        Log.e("SpamEmailDetector", "TF-IDF not initialized. Train the model first.");
        return "ham";
    }
    double spamScore = 0, hamScore = 0;
    for (int i = 0; i < tfidf.length; i++) {
        double similarity = calculateSimilarity(emailTfidf, tfidf[i]);
        if (labels.get(i).equals("spam")) {
            spamScore += similarity;
        } else {
            hamScore += similarity;
        }
    }
    // BOOST SCORE BASED ON FEATURES
    if (email.contains("http://") || email.contains("https://")) {
        spamScore += 1.5;
    }
    if (email.contains(".pdf") || email.contains(".doc") || email.contains("attachment")) {
        spamScore += 1.0;
    }
    String[] spamWords = {"winner", "congratulations", "claim", "lottery", "selected", "prize", "urgent"};
    for (String word : spamWords) {
        if (email.contains(word)) {
            spamScore += 0.5;
        }
    }
    if (email.length() > 1000) {
        spamScore += 0.5;
    }
    String result = spamScore > hamScore * 1.2 ? "spam" : "ham";
    saveEmailToFirebase(email, result, sender, subject);
    if (result.equals("spam")) {
        spamCount++;
        if (spamCount % SPAM_THRESHOLD == 0) {
            sendSpamNotification();
        }
    }
    Log.d("SpamDetection", "SPAM SCORE: " + spamScore + ", HAM SCORE: " + hamScore);
    return result;
}
```

3. Firebase Integration:

The application relies on Firebase for user data management and storing classified messages. Unit tests were written to confirm that:

Code :

- New user registrations are successfully stored in the Firebase database.

```
private void registerUser(String name, String email, String password, String phone) {
    auth.createUserWithEmailAndPassword(email, password).addOnCompleteListener(register.this, task -
> {
        if (task.isSuccessful()) {
            FirebaseUser user = auth.getCurrentUser();
            if (user != null) {
                // Reference to Firebase Realtime Database
                String userId = auth.getCurrentUser().getUid();
                databaseRef = FirebaseDatabase.getInstance().getReference("Users").child(userId);
                // Storing user data in HashMap
                HashMap<String, Object> userData = new HashMap<>();
                userData.put("name", name);
                userData.put("email", email);
                userData.put("phone", phone);b
                userData.put("password", password);
                userData.put("isActive", true); // Not recommended for security reasons
                // Save data in Realtime Database
                databaseRef.setValue(userData).addOnCompleteListener(task1 -> {
                    if (task1.isSuccessful()) {
                        Toast.makeText(register.this, "Successfully Registered!",
Toast.LENGTH_SHORT).show();
                        startActivity(new Intent(register.this, login.class));
                        finish();
                    } else {
                        Toast.makeText(register.this, "Failed to store d ata!", Toast.LENGTH_SHORT).show();
                    }
                });
            }
        } else {
            Toast.makeText(register.this, "Registration Failed: " + task.getException().getMessage(),
    Toast.LENGTH_SHORT).show();
        }
    });
    }
```

- Emails marked as spam are saved correctly under the designated node (SpamEmails).

```
private void saveEmailToFirebase(String emailText, String label, String sender, String subject) {
  String userId = FirebaseAuth.getInstance().getCurrentUser().getUid();
  DatabaseReference emailRef = FirebaseDatabase.getInstance().getReference("Users").child(userId);
  DatabaseReference targetRef = "spam".equals(label) ? emailRef.child("SpamEmails") :
emailRef.child("Emails");
  targetRef.orderByChild("subject").equalTo(subject).get().addOnCompleteListener(task -> {
    boolean isDuplicate = false;
    if (task.isSuccessful() && task.getResult().exists()) {
      for (DataSnapshot snapshot : task.getResult().getChildren()) {
        String existingSender = snapshot.child("sender").getValue(String.class);
        if (existingSender != null &&
              existingSender.trim().equalsIgnoreCase(sender.trim())) {
          Log.d("SpamDetection", "Duplicate email detected, skipping storage.");
          isDuplicate = true;
          break;
        }
      }
    }
    if (!isDuplicate) {
      String emailId = targetRef.push().getKey();
      if (emailId != null) {
        Map<String, Object> emailData = new HashMap<>();
        emailData.put("sender", sender);
        emailData.put("subject", subject);
        emailData.put("body", emailText);
        emailData.put("classification", label);
        targetRef.child(emailId).setValue(emailData);
      }
    }
  });
}
```

- The retrieval and display of emails from Firebase function as expected.

```
private void fetchSpamEmails() {
  databaseRef.addListenerForSingleValueEvent(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
      spamEmailList.clear();
      Set<String> uniqueEmails = new HashSet<>(); //     Track unique emails by "sender+subject"
```

```
            for (DataSnapshot emailSnapshot : snapshot.getChildren()) {
                String sender = emailSnapshot.child("sender").getValue(String.class);
                String subject = emailSnapshot.child("subject").getValue(String.class);
                String body = emailSnapshot.child("body").getValue(String.class);
                if (sender != null && body != null) {
                    if (subject == null) subject = "(No Subject)";
                    String uniqueKey = sender + subject; //    Unique key to prevent duplicates
                    if (!uniqueEmails.contains(uniqueKey)) {
                        uniqueEmails.add(uniqueKey);
                        spamEmailList.add(new EmailItem(sender, subject, body));
                    }
                }
            }
            if (spamEmailList.isEmpty()) {
                noSpamMessage.setVisibility(View.VISIBLE);
                recyclerView.setVisibility(View.GONE);
            } else {
                noSpamMessage.setVisibility(View.GONE);
                recyclerView.setVisibility(View.VISIBLE);
                emailAdapter.notifyDataSetChanged();
            }
        }
        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Toast.makeText(getActivity(), "Failed to load spam emails", Toast.LENGTH_SHORT).show();
        }
    });
}
```

Test Case :

Test Case ID	Test Scenario	Test Steps	Expected Result
FB_TC_01	Save user data on registration	Register with valid name, email, phone, and password	User data is saved in **Firebase Realtime Database** under /Users/<uid>
FB_TC_02	Save classified spam message	After email classification as spam, store it in Firebase	Spam email stored under /Users/<uid>/SpamEmails
FB_TC_03	Prevent storing ham messages	Classify ham message	Ham emails **not stored** under SpamEmails
FB_TC_04	Fetch spam emails correctly	Navigate to GalleryFragment	Previously stored spam emails are displayed correctly
FB_TC_05	Ensure correct Firebase path usage	Store email under user ID	Data is stored under correct UID path
FB_TC_06	User login loads data	Login with valid credentials	Associated user data (like name, email) is fetched from Firebase
FB_TC_07	Handle Firebase write failure	Disconnect internet before saving spam email	App shows error Toast message: "Failed to store data!"
FB_TC_08	Handle Firebase read failure	Disconnect internet before fetching spam emails	App handles error gracefully with error message or fallback

5. **Notification & UI Behavior:**
Tests were also conducted to verify user notifications when a message is classified as spam, or when actions such as sending, receiving, or classifying emails are performed. In particular, the app displays a toast message when a user attempts to send a message that is detected as spam, thereby preventing the message from being sent.

Code :

```
private static void createNotificationChannel(Context context) {
   if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
      CharSequence name = "Spam Alerts";
      String description = "Notifies when spam emails are detected";
      int importance = NotificationManager.IMPORTANCE_HIGH;
      NotificationChannel channel = new NotificationChannel(CHANNEL_ID, name, importance);
      channel.setDescription(description);
      NotificationManager notificationManager = context.getSystemService(NotificationManager.class);
      notificationManager.createNotificationChannel(channel);
   }
}
```

Test Case :

Test Case ID	Test Scenario	Test Steps	Expected Result
UI_TC_01	Show spam warning on compose	Compose a message with spam keywords and click send	Toast: "Message detected as spam. Sending blocked."
UI_TC_02	Allow sending ham message	Compose a normal message and click send	Message sent successfully; Toast: "Message sent."
UI_TC_03	Show classification notification	Receive a new email	App classifies message and optionally shows classification result
UI_TC_04	Block spam message sending	Attempt to send spam message	Email is not sent; toast is shown
UI_TC_05	Display toast on login failure	Login with invalid credentials	Toast: "Authentication failed…" is shown
UI_TC_06	Display toast on Firebase failure	Turn off internet and try saving spam email	Toast: "Failed to store data!" is shown
UI_TC_07	Toast on registration success	Register a new user with valid credentials	Toast: "Successfully Registered!" is displayed
UI_TC_08	Toast on logout	Click logout in Navigation Drawer	User is logged out, and toast (if implemented) confirms action

The unit tests were implemented using the JUnit framework for Java, simulating real-world conditions and edge cases. This helped in identifying issues early in the development process and ensured that each module performed its intended function before being integrated into the complete system.

4.2.2 Integration Testing

Integration testing involved validating the interaction between various components of the spam email detection mobile application to ensure that they function cohesively. These tests were conducted after unit testing to check how different modules work together as part of a complete system.

The primary goal was to verify that the following workflows performed as expected:

- User Authentication Flow: The login and registration modules were tested to confirm that valid users could access their inbox and that invalid credentials triggered appropriate error messages.
- Email Fetching and Display: Once authenticated, the system fetched emails from Gmail using IMAP and displayed them in the user's inbox. Integration tests confirmed that the email list was accurately loaded and updated dynamically.
- Spam Detection and Classification: The integration of the spam detection algorithm with the email processing system was tested by inputting various sample emails. If a message was identified as spam, the system moved it to a dedicated "Trash" folder and stored it in Firebase under the SpamEmails node.
- Firebase Integration: Tests confirmed that spam emails were stored in Firebase correctly, including their sender, subject, and cleaned content. It was also verified that legitimate (ham) emails remained in the inbox and were not mistakenly flagged.
- User Interface and Model Communication: Integration tests confirmed that the communication between the machine learning model (SpamEmailDetector), backend (Firebase), and frontend (UI) was seamless and responsive to user actions.
- User Notification on Spam: Users were properly notified with toast messages or blocked when attempting to send spam content using the Compose feature.

Test Case :

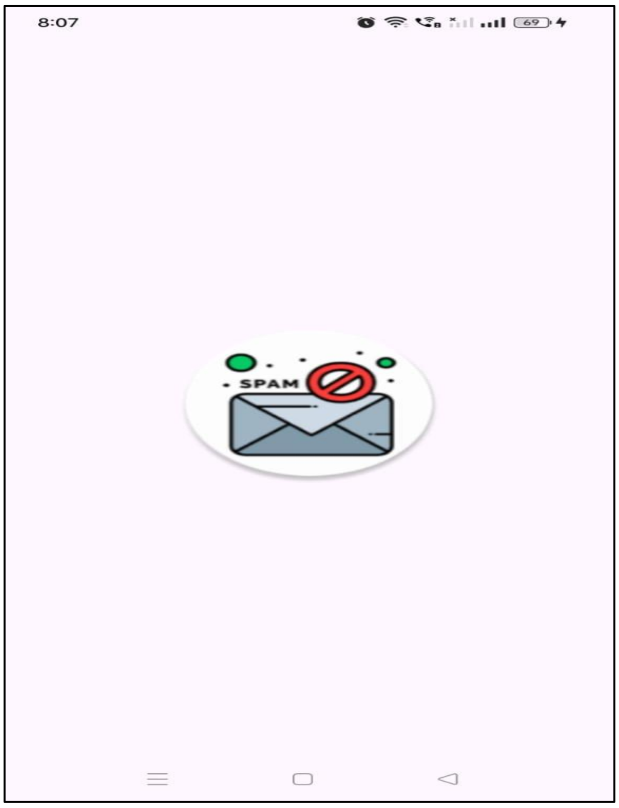
Test Case ID	Integration Scenario	Test Steps	Expected Result	Actual Result
INT_TC_01	User Authentication Flow	Register and login using valid credentials	User is redirected to inbox screen	Pass
INT_TC_02	User Authentication Error Handling	Try to login with invalid credentials	Error toast displayed: "Authentication failed…"	Pass
INT_TC_03	Email Fetching After Login	Login with valid credentials and wait for	Emails are fetched via IMAP and displayed in inbox	Pass

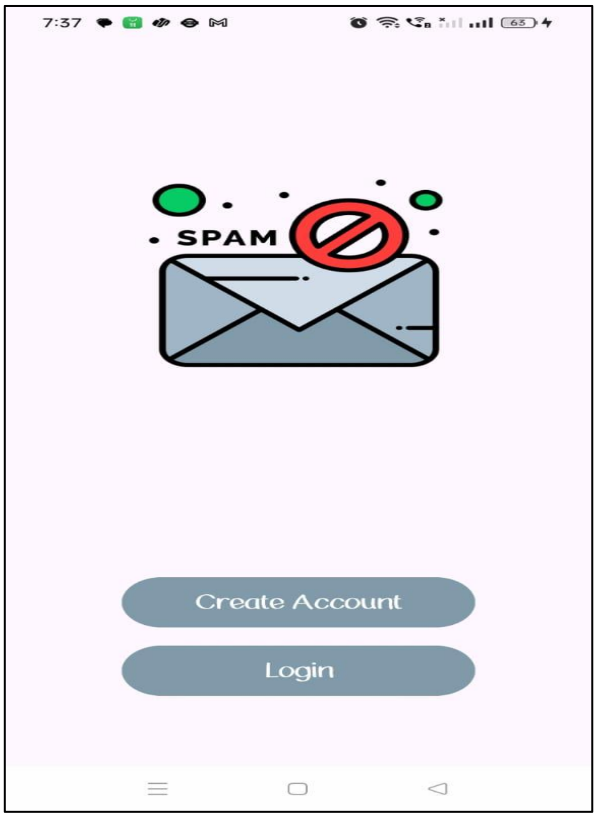
		inbox load		
INT_TC_04	Email Display Update	Scroll through inbox	Email list loads dynamically in batches	Pass
INT_TC_05	Spam Detection Flow	Receive emails with spam keywords or suspicious links	Spam emails are detected and moved to Trash folder	Pass
INT_TC_06	Firebase Storage for Spam	After detection, check Firebase under SpamEmails node	Sender, subject, and cleaned body stored correctly	Pass
INT_TC_07	Legitimate Email Handling	Receive ham (non-spam) email	Email stays in inbox and is not moved to Trash	Pass
INT_TC_08	Compose Block on Spam	Try sending a spam-like email using Compose	Toast shown and sending blocked	Pass
INT_TC_09	UI + Model Interaction	User composes & sends a clean email	Message passes SpamEmailDetector, gets sent successfully	Pass
INT_TC_10	System Flow Stability	Perform full flow: login → inbox → spam classify → compose	App remains responsive and modules interact properly	Pass

Through this testing phase, it was ensured that the **user experience remained uninterrupted**, and all the core features—from login to classification—interacted correctly and robustly as a single application.

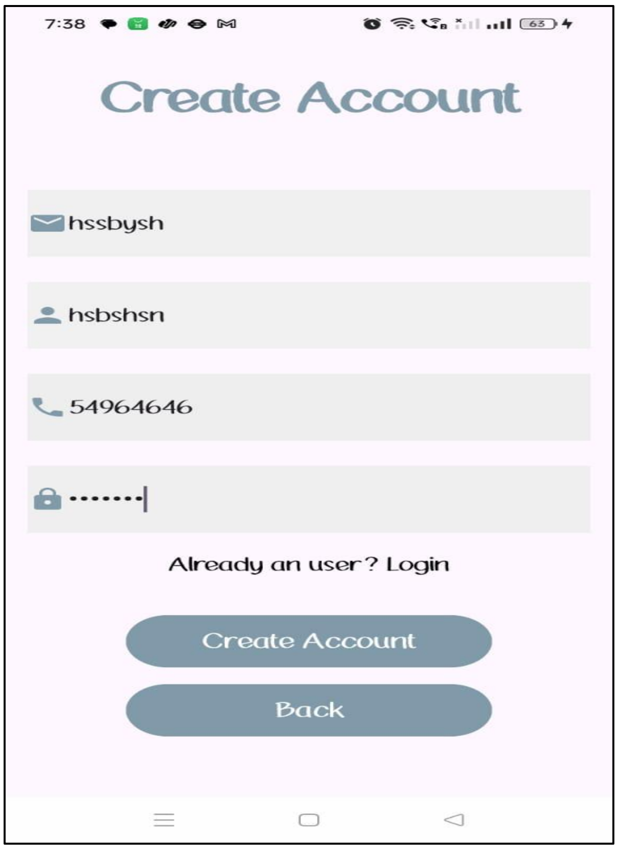
Chapter 5 : Result and Discussions

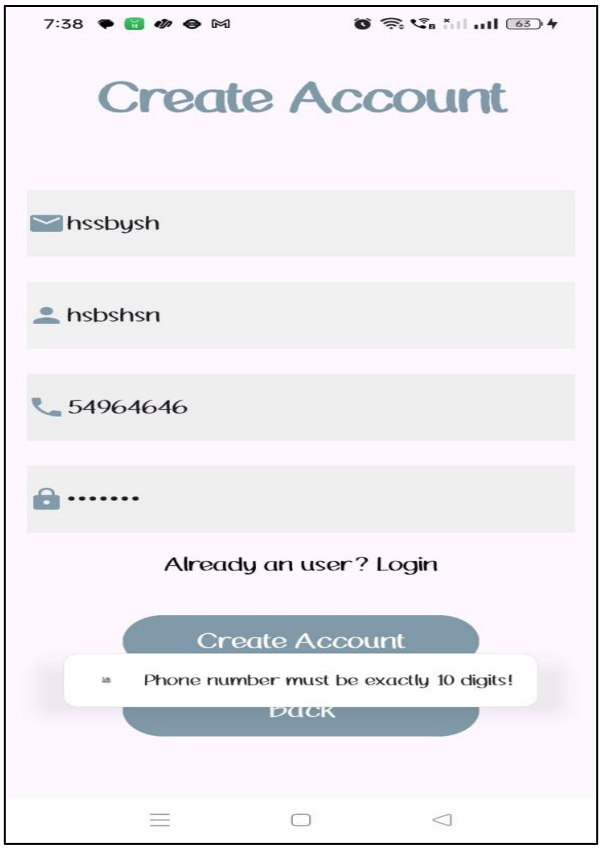
1. The splash screen is displayed when the user opens the app for the first time.





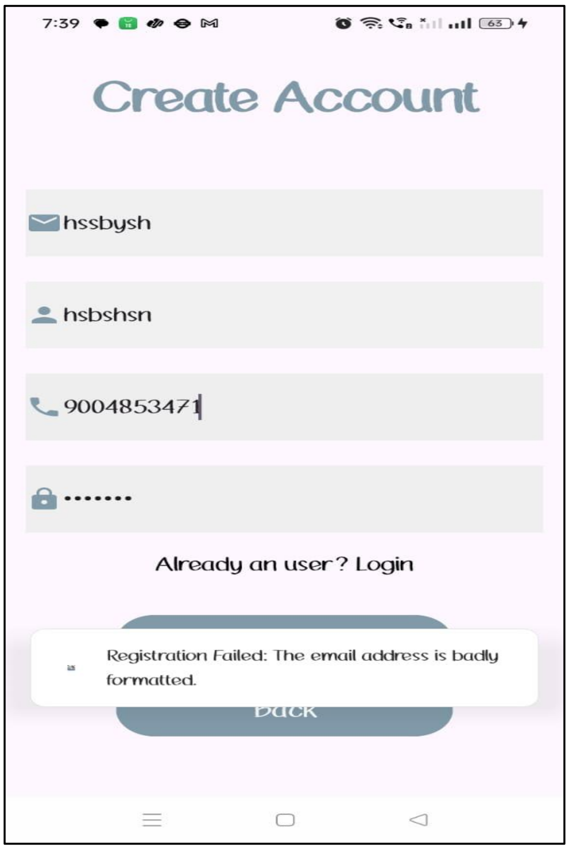
2. The user can choose to register a new account by clicking the "Create Account" button.





Phone number must be exactly 10 digits!

3. On the registration page, the user enters their name, email, phone number, and password.


7:39

Create Account

hssbysh

hsbshsn

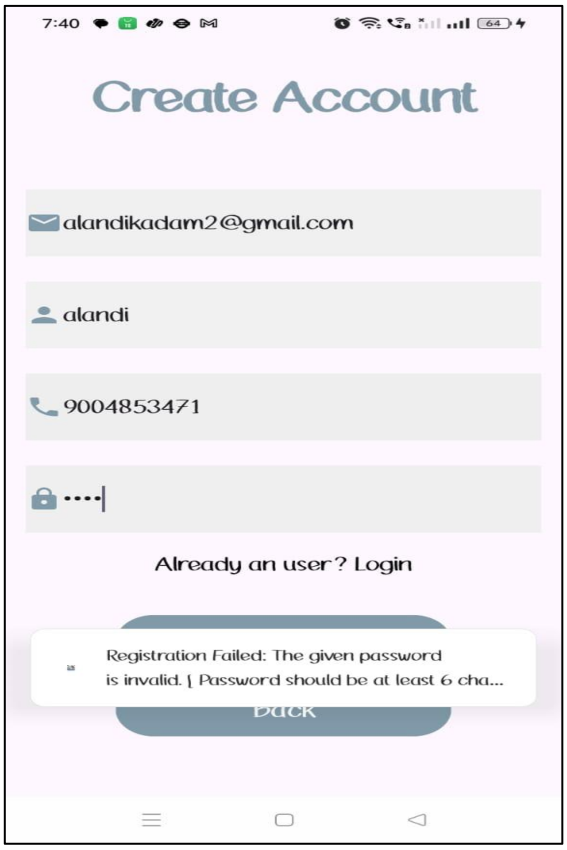
9004853471

•••••••

Already an user? Login

Registration Failed: The email address is badly formatted.

Back


7:40

Create Account

alandikadam2@gmail.com

alandi

9004853471

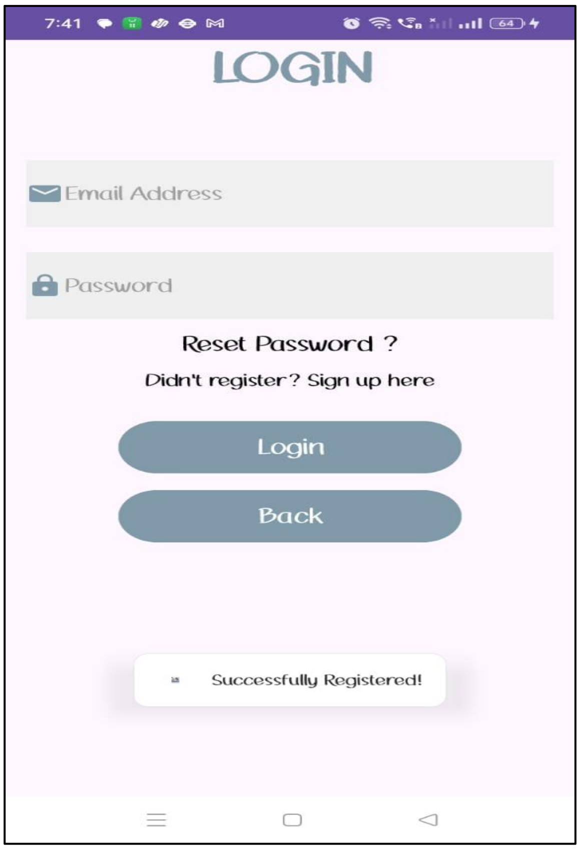
••••

Already an user? Login

Registration Failed: The given password is invalid. [Password should be at least 6 cha...

Back

4. After successful registration, the user is redirected to the login page.


7:41

LOGIN

Email Address

Password

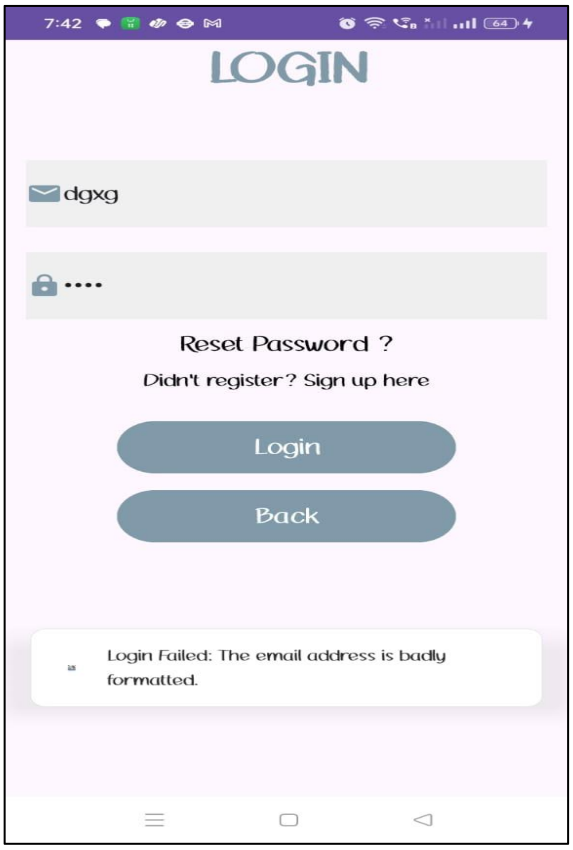
Reset Password ?

Didn't register? Sign up here

Login

Back

Successfully Registered!


7:42

LOGIN

dgxg

••••

Reset Password ?

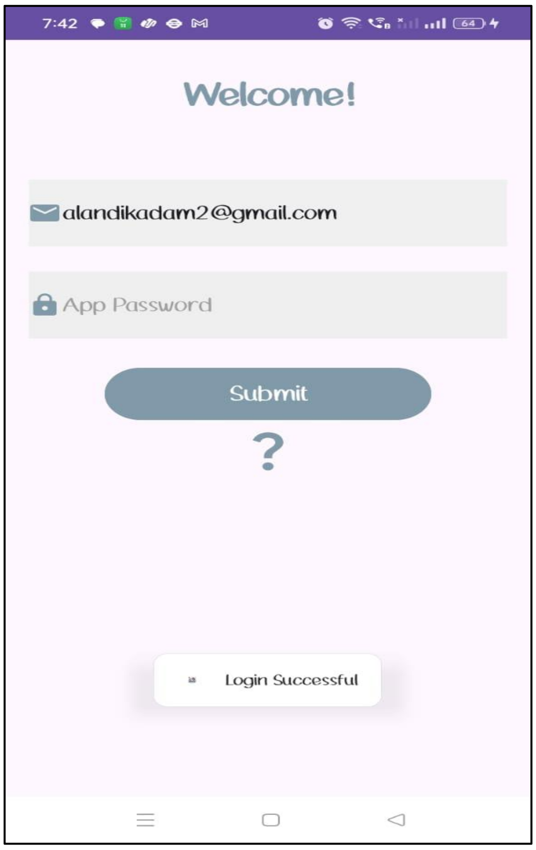
Didn't register? Sign up here

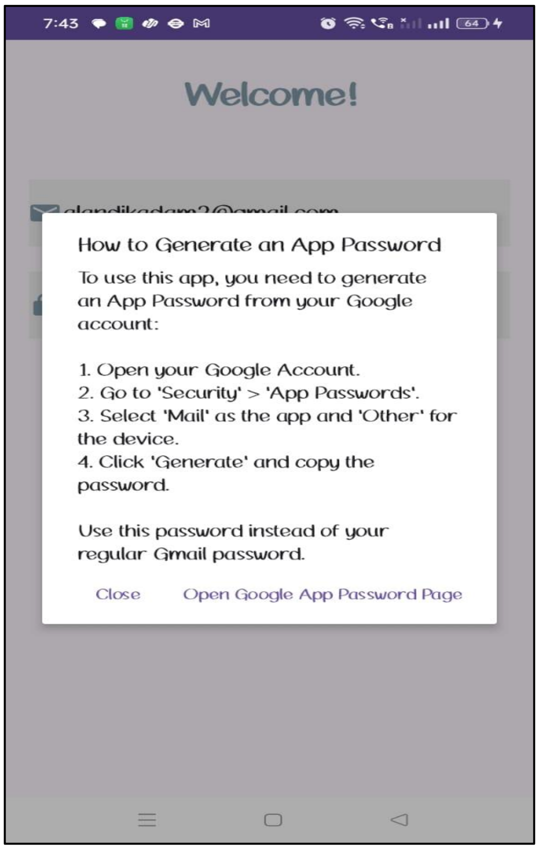
Login

Back

Login Failed: The email address is badly formatted.

5. If the credentials are correct, the user is authenticated and redirected to the Home page





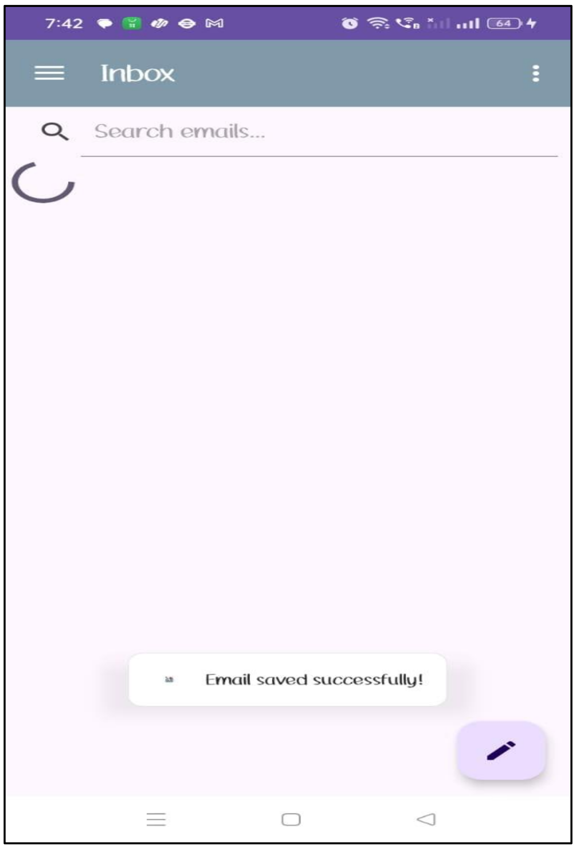
How to Generate an App Password

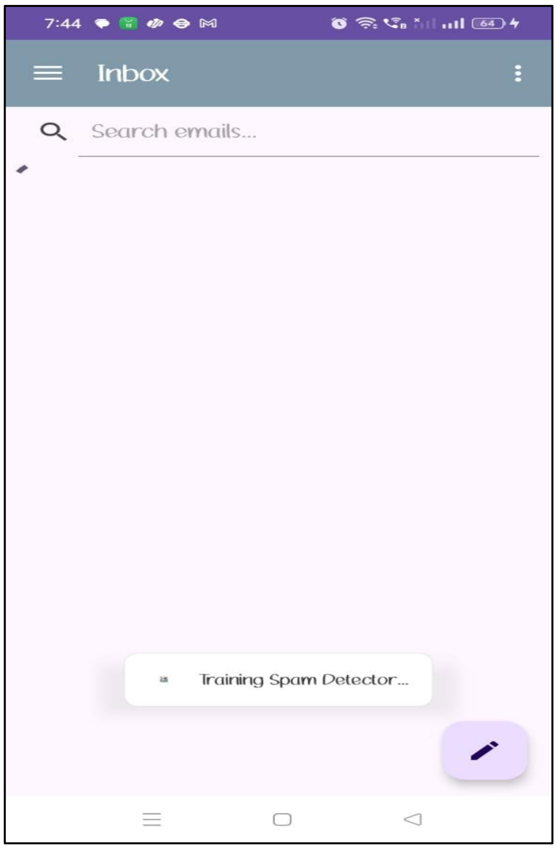
To use this app, you need to generate an App Password from your Google account:

1. Open your Google Account.
2. Go to 'Security' > 'App Passwords'.
3. Select 'Mail' as the app and 'Other' for the device.
4. Click 'Generate' and copy the password.

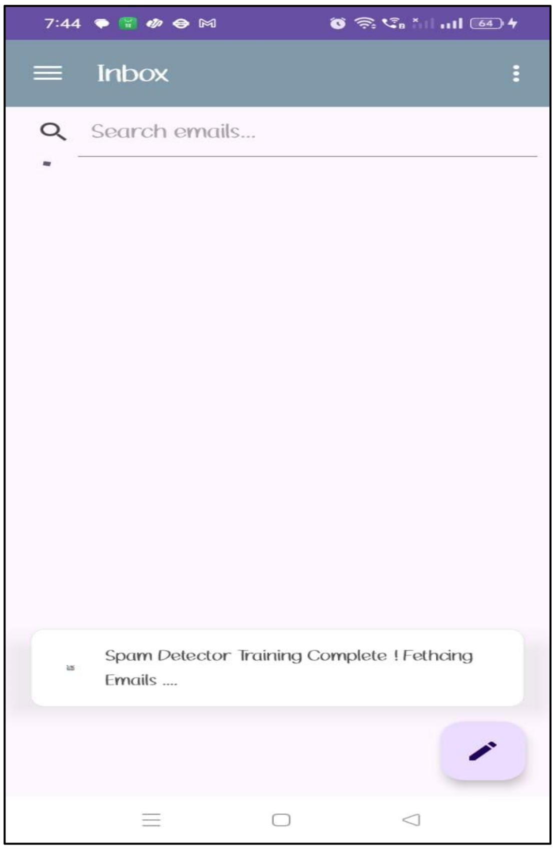
Use this password instead of your regular Gmail password.

6. For each email fetched, the app uses the SpamEmailDetector to classify it as spam or not.

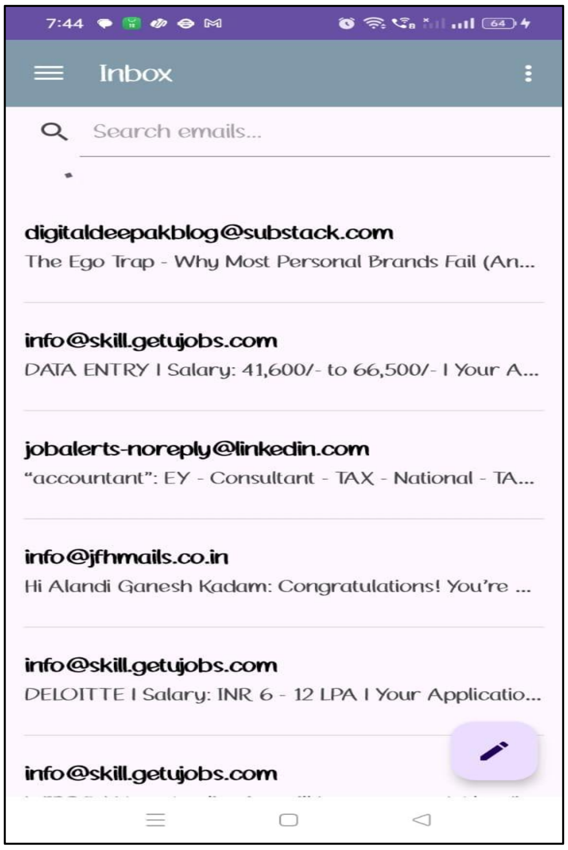




7. The app fetches emails from the user's Gmail inbox using IMAP and displays them in batches.



Spam Detector Training Complete ! Fethcing Emails ....



digitaldeepakblog@substack.com
The Ego Trap - Why Most Personal Brands Fail (An...

info@skill.getujobs.com
DATA ENTRY I Salary: 41,600/- to 66,500/- I Your A...

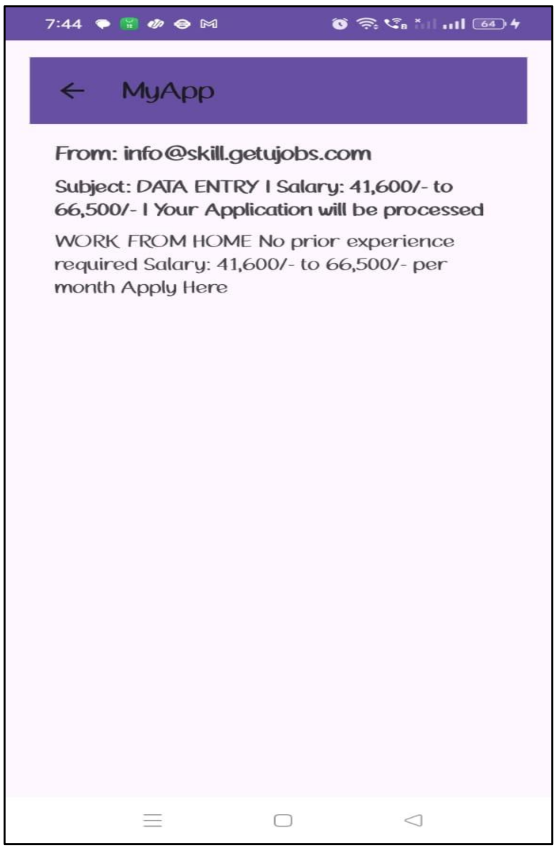
jobalerts-noreply@linkedin.com
"accountant": EY - Consultant - TAX - National - TA...

info@jfhmails.co.in
Hi Alandi Ganesh Kadam: Congratulations! You're ...

info@skill.getujobs.com
DELOITTE I Salary: INR 6 - 12 LPA I Your Applicatio...

info@skill.getujobs.com

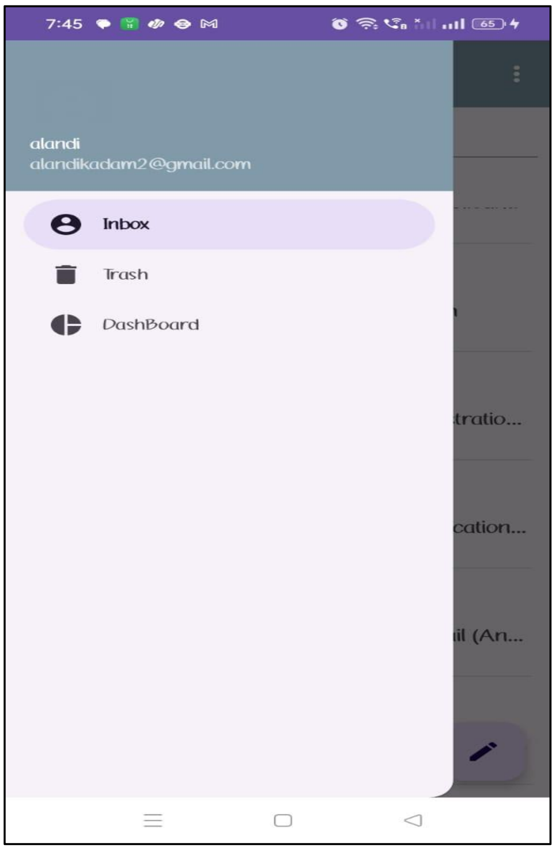
8. Spam emails are moved to the Trash folder in Gmail and stored in Firebase under the SpamEmails node.



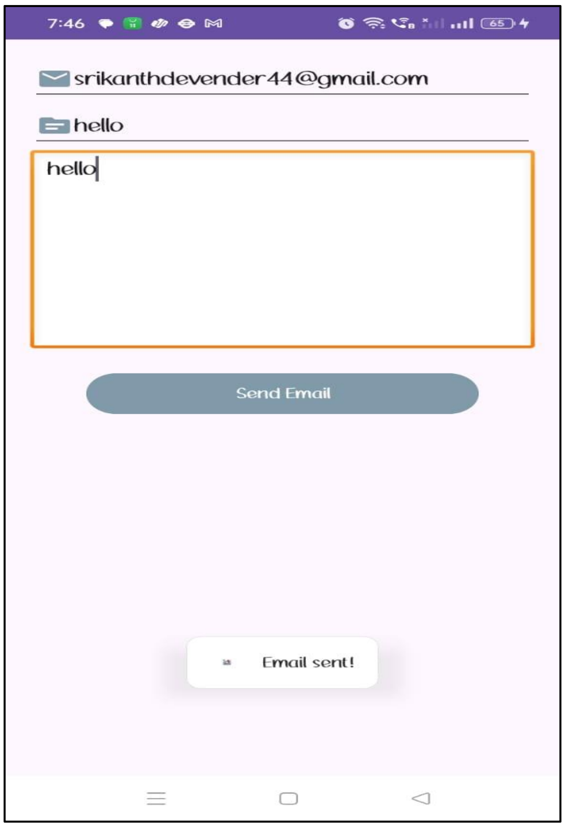
From: info@skill.getujobs.com

Subject: DATA ENTRY I Salary: 41,600/- to 66,500/- I Your Application will be processed

WORK FROM HOME No prior experience required Salary: 41,600/- to 66,500/- per month Apply Here



9. The user can compose a new email from the Compose screen.If the message is detected as spam, a toast message is displayed, and the email is not sent.

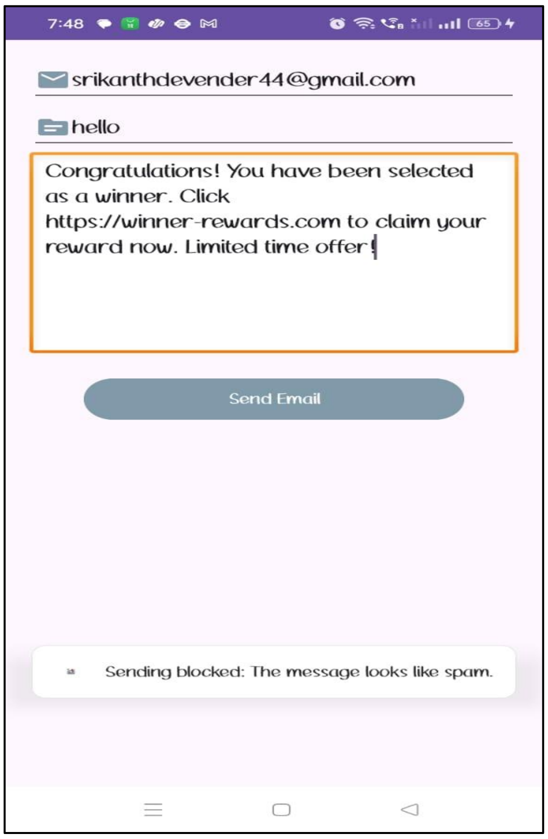


srikanthdevender44@gmail.com

hello

hello

Email sent!



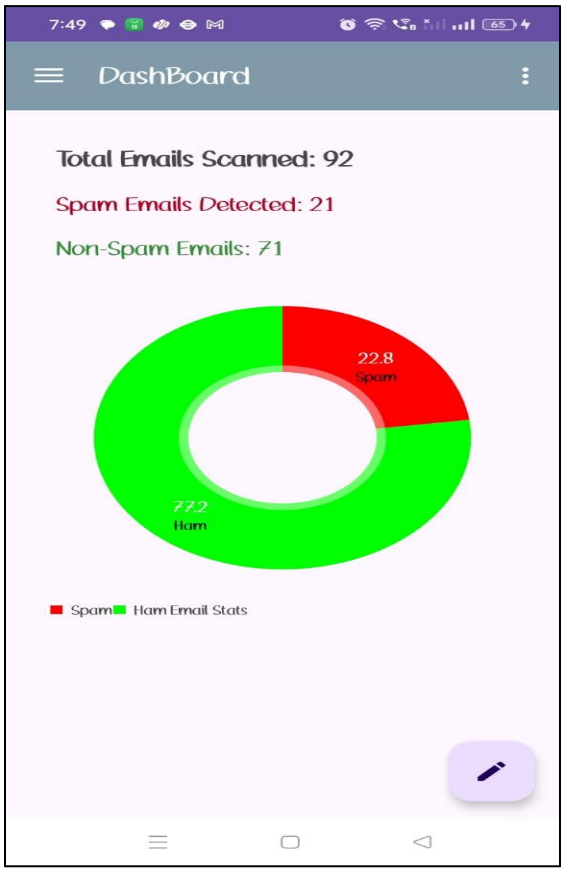
srikanthdevender44@gmail.com

hello

Congratulations! You have been selected as a winner. Click https://winner-rewards.com to claim your reward now. Limited time offer!

Sending blocked: The message looks like spam.

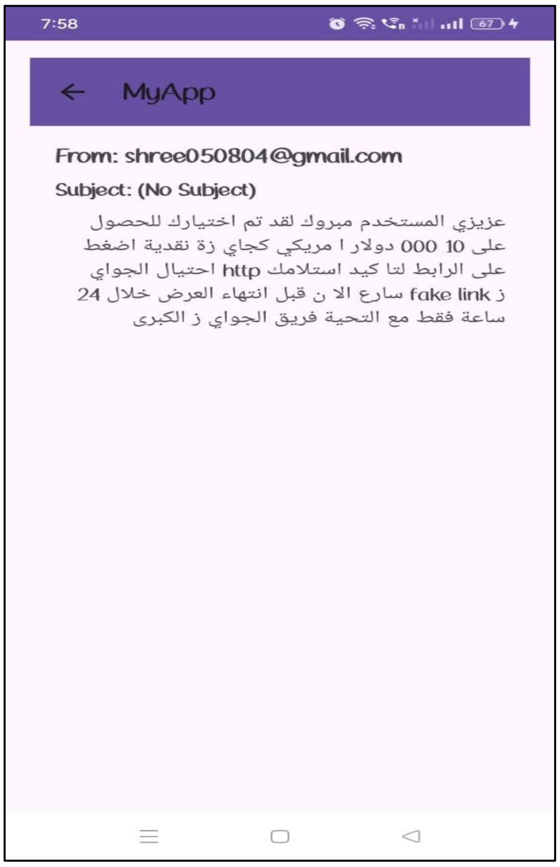
10. In the Dashboard the user can view statistical summaries of spam detection, including counts and Trends.Can detect spam message in multiple language.



Total Emails Scanned: 92

Spam Emails Detected: 21

Non-Spam Emails: 71

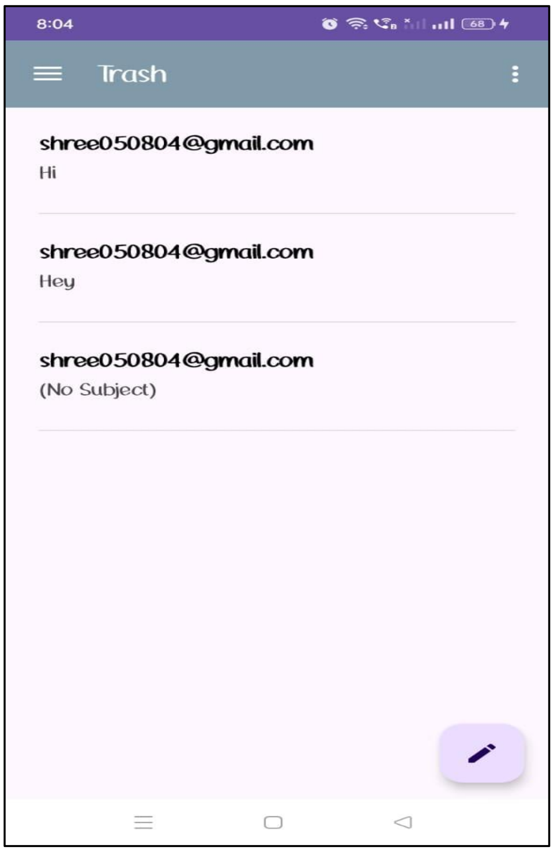


From: shree050804@gmail.com

Subject: (No Subject)

عزيزي المستخدم مبروك لقد تم اختيارك للحصول على 10 000 دولار ا مريكي كجاي زة نقدية اضغط على الرابط لتا كيد استلامك http احتيال الجواي ز fake link سارع الا ن قبل انتهاء العرض خلال 24 ساعة فقط مع التحية فريق الجواي ز الكبرى

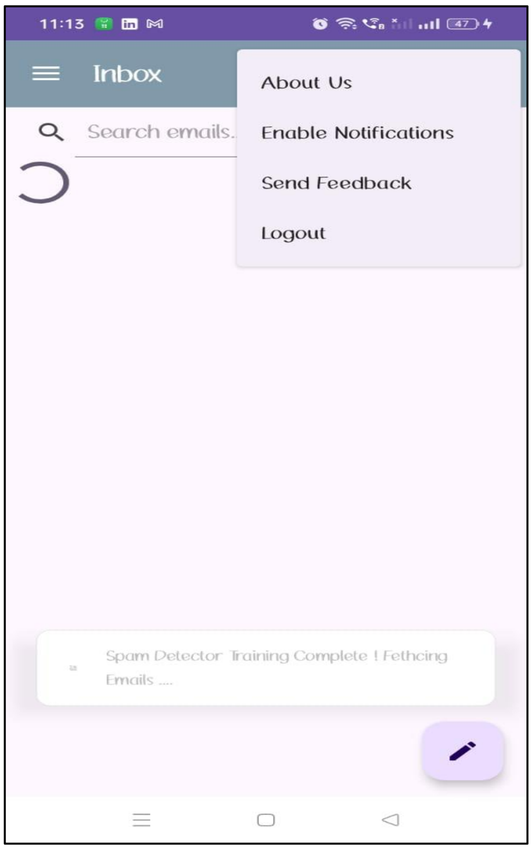
.



shree050804@gmail.com
Hi

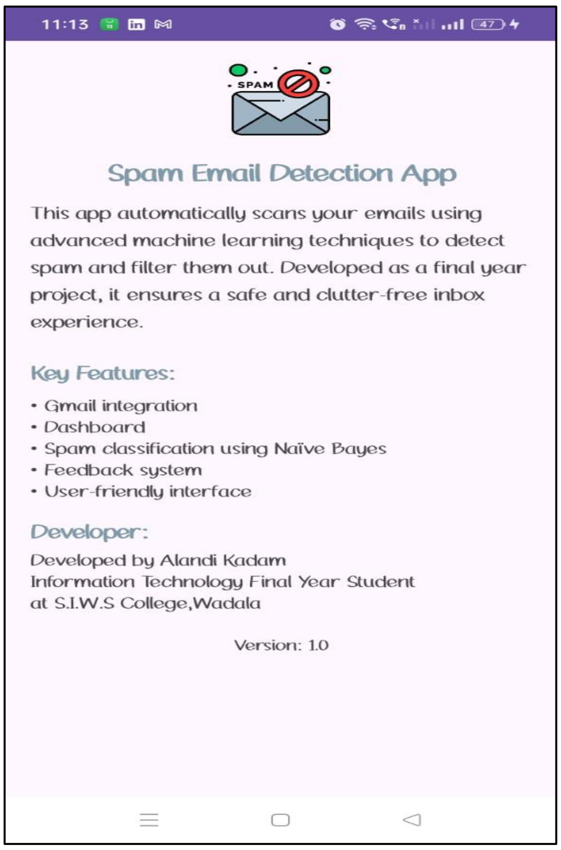
shree050804@gmail.com
Hey

shree050804@gmail.com
(No Subject)



Spam Detector Training Complete ! Fethcing Emails ....

11. Here added some features . user can see about us , enable disable notification , gives feedback and logout from current account .



Spam Email Detection App

This app automatically scans your emails using advanced machine learning techniques to detect spam and filter them out. Developed as a final year project, it ensures a safe and clutter-free inbox experience.

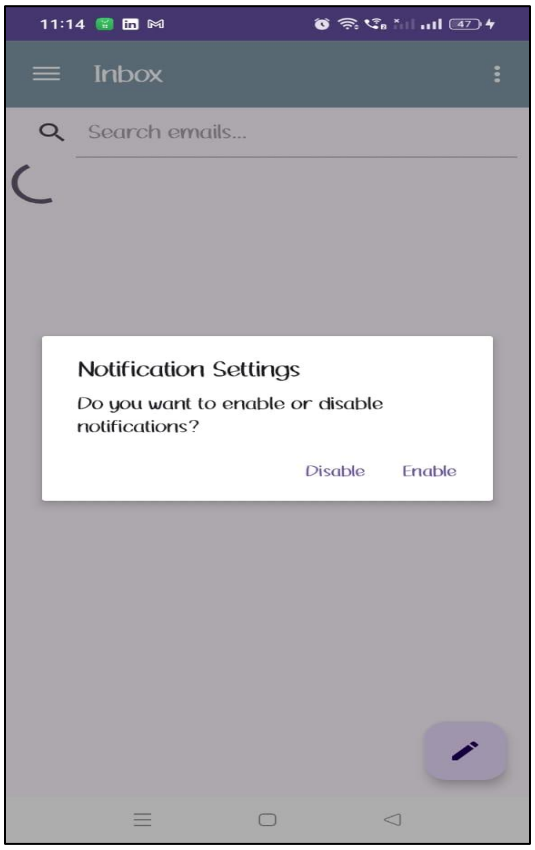
Key Features:

- Gmail integration
- Dashboard
- Spam classification using Naïve Bayes
- Feedback system
- User-friendly interface

Developer:

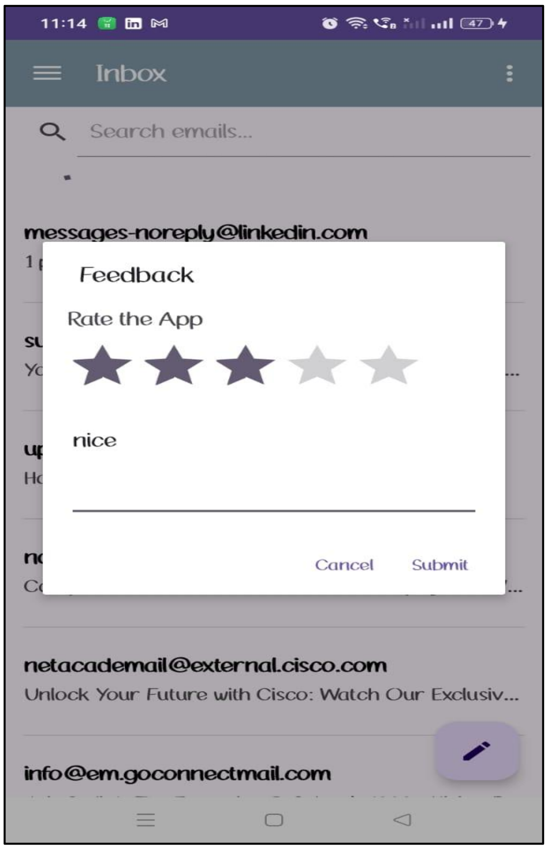
Developed by Alandi Kadam
Information Technology Final Year Student
at S.I.W.S College,Wadala

Version: 1.0



Notification Settings

Do you want to enable or disable notifications?


Inbox

Search emails...

messages-noreply@linkedin.com

Feedback

Rate the App

★★★☆☆

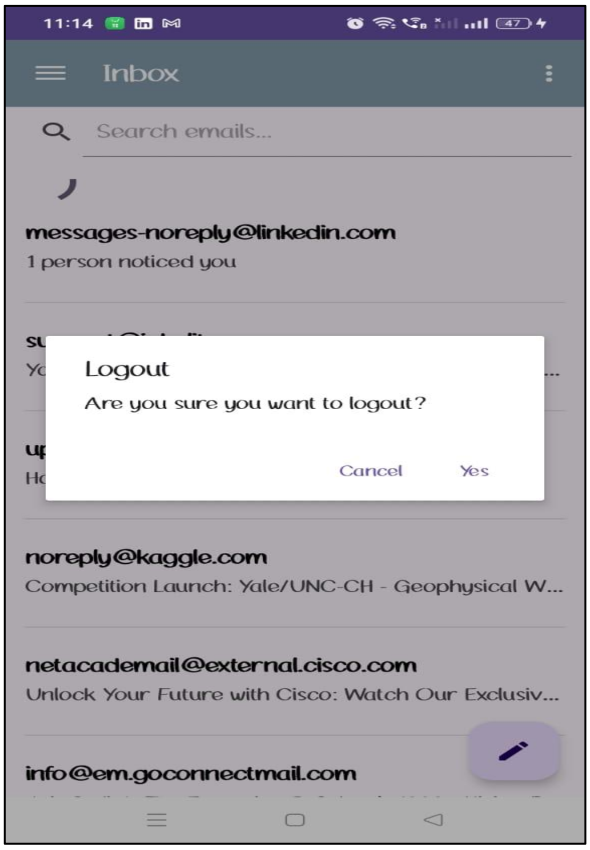
nice

Cancel Submit

netacademail@external.cisco.com

Unlock Your Future with Cisco: Watch Our Exclusiv...

info@em.goconnectmail.com


Inbox

Search emails...

messages-noreply@linkedin.com

1 person noticed you

Logout

Are you sure you want to logout?

Cancel Yes

noreply@kaggle.com

Competition Launch: Yale/UNC-CH - Geophysical W...

netacademail@external.cisco.com

Unlock Your Future with Cisco: Watch Our Exclusiv...

info@em.goconnectmail.com

Chapter 6: Conclusion and Future Work

6.1 Conclusion

This project successfully delivers an Android-based spam email detection system that empowers users to identify and manage spam messages using machine learning techniques. By integrating Gmail IMAP services and a custom-trained Naïve Bayes classifier enhanced with TF-IDF and cosine similarity, the app offers real-time spam filtering and email classification. The application also stores spam messages in Firebase, enabling users to track and review detected spam, while ensuring efficient data retrieval through batch loading and background processing.

6.1.1 Significance of the System

- Enhanced Email Security: The system provides an added layer of protection by flagging and preventing spam emails from being sent or received.

- User-Friendly Interface: With a clean and intuitive UI, users can navigate between inbox and spam emails effortlessly.

- Integration with Firebase: Enables real-time storage, user authentication, and retrieval of emails across sessions and devices.

6.2 Limitations of the System

- Static Model Training: The spam detection model does not learn dynamically from new spam samples or user feedback.

- Limited Language Support: Though the classifier handles multiple languages, accuracy may reduce for complex or mixed-language emails.

- Basic Spam Detection: Only text-based features and metadata are used; no deep learning or advanced NLP techniques are implemented.

- IMAP Dependency: The system currently supports only IMAP email providers (like Gmail, Yahoo, Outlook), excluding POP3 services.

6.3 Future Scope of the Project

- Real-Time Learning: Implement feedback-based learning to allow the model to improve with user actions (e.g., marking messages as spam or ham).

- Attachment and Link Analysis: Extend the classifier to analyze links and attachments, improving detection of phishing or malware emails.

- Advanced Visualization: Introduce detailed statistics in the dashboard, including spam trends, top spam senders, and time-based analysis.

- Voice Commands and AI Assistant**:** Integrate voice-based querying and smart suggestions to enhance accessibility.

- Offline Classification: Allow spam detection on emails downloaded offline, making the app usable without continuous internet access.

- Admin Panel: Add a backend admin interface to manage reported spam trends and global updates to spam patterns or signature databases.

Chapter 7: References

- Zhang, H., & Lee, W. (2017). "An Adaptive Approach to Spam Filtering." Journal of Computer Science and Technology, 32(2), 323-334. DOI: 10.1007/s11390-017-1736-5.
- Bhatia, S., & Bansal, P. (2018). "Machine Learning Techniques for Email Spam Filtering: A Survey." International Journal of Computer Applications, 182(2), 13-18. DOI: 10.5120/ijca2018917761.
- Rabbani, M., & Sultana, N. (2020). "Spam Detection Using Machine Learning: A Review." *International Journal of Scientific Research in Computer Science, Engineering and Information
- Zhang, L., Zhu, J., & Yao, T. (2004). An evaluation of statistical spam filtering techniques. ACM Transactions on Asian Language Information Processing, 3(4), 243–269. https://doi.org/10.1145/1037811.1037813
- Rashid, A., Ghazi, H.A., & Khan, J.A. (2017). Spam email detection using a combination of text-based and rule-based techniques. ICIST 2017: International Conference on Information and Software Technologies.
- Google Firebase Documentation. (2023). Firebase Documentation. Google Cloud. Retrieved from https://firebase.google.com/docs