Software Design Document

Echo

Version 1.0

Prepared by

Ashwin Binu Abraham
Deepak P
Salihu Ahamed
Fathima Nooha Kottangodan

TKM College of Engineering

3rd April 2023

Table of Contents

1. Introduction 4

1.1 Purpose 4
1.2 Scope 4
1.3 Intended audience 5
1.4 References 5

2. System Architecture Design 6

2.1 Description 6
2.2 System Architecture 6

3. Application Architecture Design 7

4. GUI Design 8

4.1 User Interface Design 8
4.2 Visual Design 14
4.3 Navigation Design 14

5. API Design 15

5.1 Flutter API 15
5.2 Django API 15
5.3 NLP API 15

6. Technological Stacks 16

7. Component Design 17

8. Algorithm Design 18

9. Data Design 20

10. Error Handling Design 22

11. Performance Design 23

12. Security Design 25

13. Testing Design 26

13.1 Testing Approach 26
13.2 Testing Scope 26
13.3 Test Cases 26
13.4 Testing Tools 29

13.3 Test Environment 30
13.3 Test Data 30
13.3 Acceptance Criteria 31

13. Maintenance Design 32

14. Deployment Design 34

Appendix A: Record of Changes 35

Appendix B: Acronyms 36

Appendix C: Referenced Documents 37

List of Figures

Figure 1 - Architectural Overview 06

Figure 2 - Splash Screen GUI 08

Figure 3 - Login Screen GUI 09

Figure 4 - Home Screen GUI 10

Figure 5 - Compose Screen 11

Figure 6 - Read Email Screen 12

Figure 7 - Draft Screen 13

Figure 8 - Settings Screen 13

Figure 9 - Component Design 17

List of Tables

Table 1 - Record of Changes 35

Table 2 - Acronyms 36

Table 4 - Referenced Documents 37

1. Introduction

This Software Design Document (SDD) is intended to provide a comprehensive overview of the design and architecture of a software system. The document is aimed at developers, architects, and stakeholders involved in the development of the system. It outlines the key design decisions, architectural patterns, and technologies used to build the system.

The SDD covers various aspects of the software design, including system architecture design, application architecture design, GUI design, API design, and technology stack. The document provides detailed descriptions of the design decisions and provides an overview of the system architecture, the application architecture, and the user interface design. Additionally, it provides details about the API design and the technologies used to build the system.

This document aims to serve as a reference guide for the development team throughout the development process. It will help to ensure that the team remains aligned with the design goals and can make informed decisions regarding the implementation of the system.

1.1 Purpose

The purpose of the Software Design Document (SDD) is to provide a comprehensive overview of the design of the software system being developed. It serves as a guide for the developers, testers, and stakeholders involved in the project, and provides a common understanding of the system architecture, application architecture, GUI design, API design, database design, and technology stack. The document outlines the technical aspects of the software and the design decisions that were made during the development process, ensuring that the software meets the specified requirements and performs as expected. The SDD also helps to identify potential issues and risks that may arise during the development and implementation phases, enabling proactive measures to be taken to mitigate them. Overall, the SDD serves as a critical reference document for the development team throughout the software development lifecycle.

1.2 Scope

The scope of this Software Design Document (SDD) is to provide a comprehensive overview of the design and architecture of the proposed application. It covers the system architecture design, application architecture design, GUI design (mockups), API design, and the technology stack that will be used to develop the application. The SDD aims to provide a clear understanding of the design and architecture of the application to the development team and stakeholders involved in the project. It will also serve as a reference guide for the development team throughout the development process. The SDD is intended to be a living document and will be updated as needed throughout the project lifecycle.

1.3 Intended Audience

The intended audience for this SDD includes software developers, College Professors, and other stakeholders involved in the software development process. This document is intended to provide a detailed design of the system architecture, application architecture, GUI design, API design, and technology stack, as well as any other relevant technical details related to the development of the software application.

The audience can review the overall design and technical description, and focus on specific components such as the system architecture, application architecture, GUI design, API design, and technology stack. Any concerns or areas of interest that could impact the development or evaluation of the software application can be noted for further discussion and clarification

1.4 References

- IEEE Standards Association. IEEE Std 1016-2009, IEEE Standard for Information Technology--Systems Design--Software Design Descriptions. IEEE, 2009.
- Martin, Robert C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017.

2. System Architecture Design

2.1 Description

The architecture consists of three main tiers: the Presentation Tier, the Functionality Tier, and the Data Storage Tier. The Presentation Tier is responsible for the user interface and screens, which are developed using Flutter, and the voice command plugin. The Functionality Tier is responsible for the email service, which handles communication with the Gmail API and Outlook API to perform email operations such as sending, receiving, and reading emails, as well as the email processor, which processes received emails and stores them locally in the device storage for offline access. The Functionality Tier also includes the Flutter plugin for enabling voice commands. The Data Storage Tier is responsible for locally storing the fetched emails for offline access, and the emails are stored in the device's local storage.

These three tiers work together to provide the complete functionality of the system, allowing visually impaired people to perform basic email operations through voice commands using the mobile app

2.2 Architecture

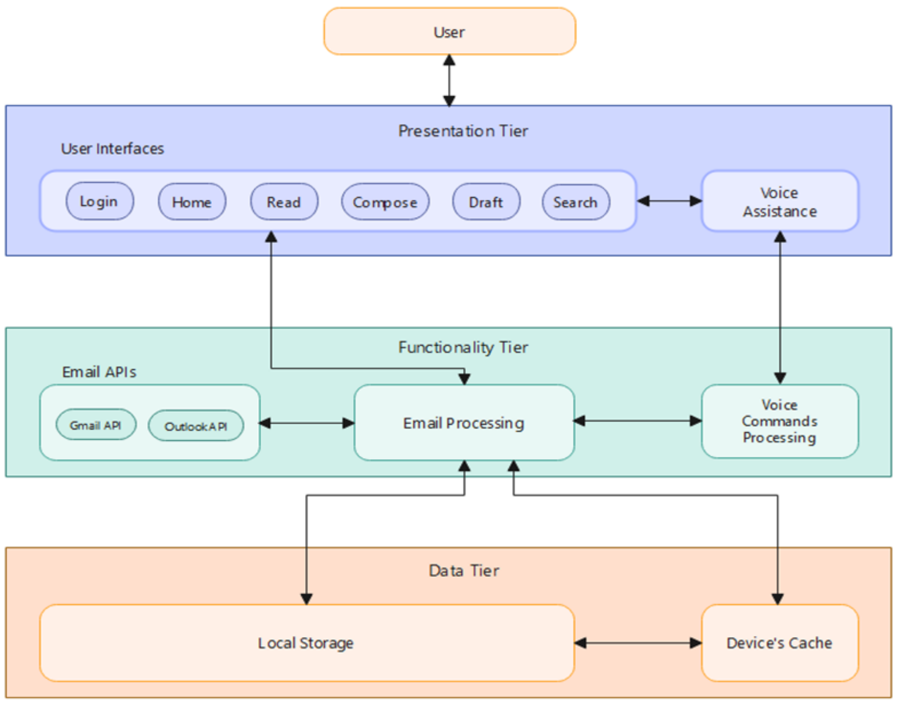


Figure 1.0: Architectural Overview

3. Application Architecture Design

The Echo app will be a mobile email client designed for visually impaired users. The application architecture will consist of two main components: a frontend built using the Flutter framework, and a backend built using the Django web framework.

The frontend will handle the user interface and the voice command processing. The interface will be designed with accessibility in mind, with high-contrast and large text options available. The voice command processing will be implemented using Flutter plugins.

The backend will handle the email operations and will integrate with the Gmail and Outlook APIs for email retrieval, sending, and management. The backend will also implement the authentication flow for both Gmail and Outlook.

Overall, the application architecture is designed to provide a smooth and accessible email experience for visually impaired users, utilizing the latest technologies for mobile app development and API integration

4. GUI Design

4.1 User Interface Design

a. Splash Screen
Displaying the application logo and name for 1.5 seconds.

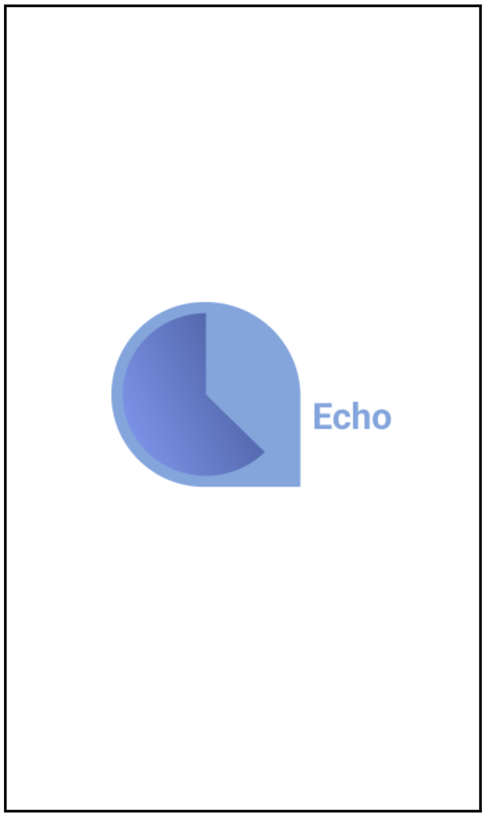


Figure 2.0: Splash Screen GUI

b. Login Screen

A voice-over provides instructions on how to login. The text boxes are filled as per the voice input from users.

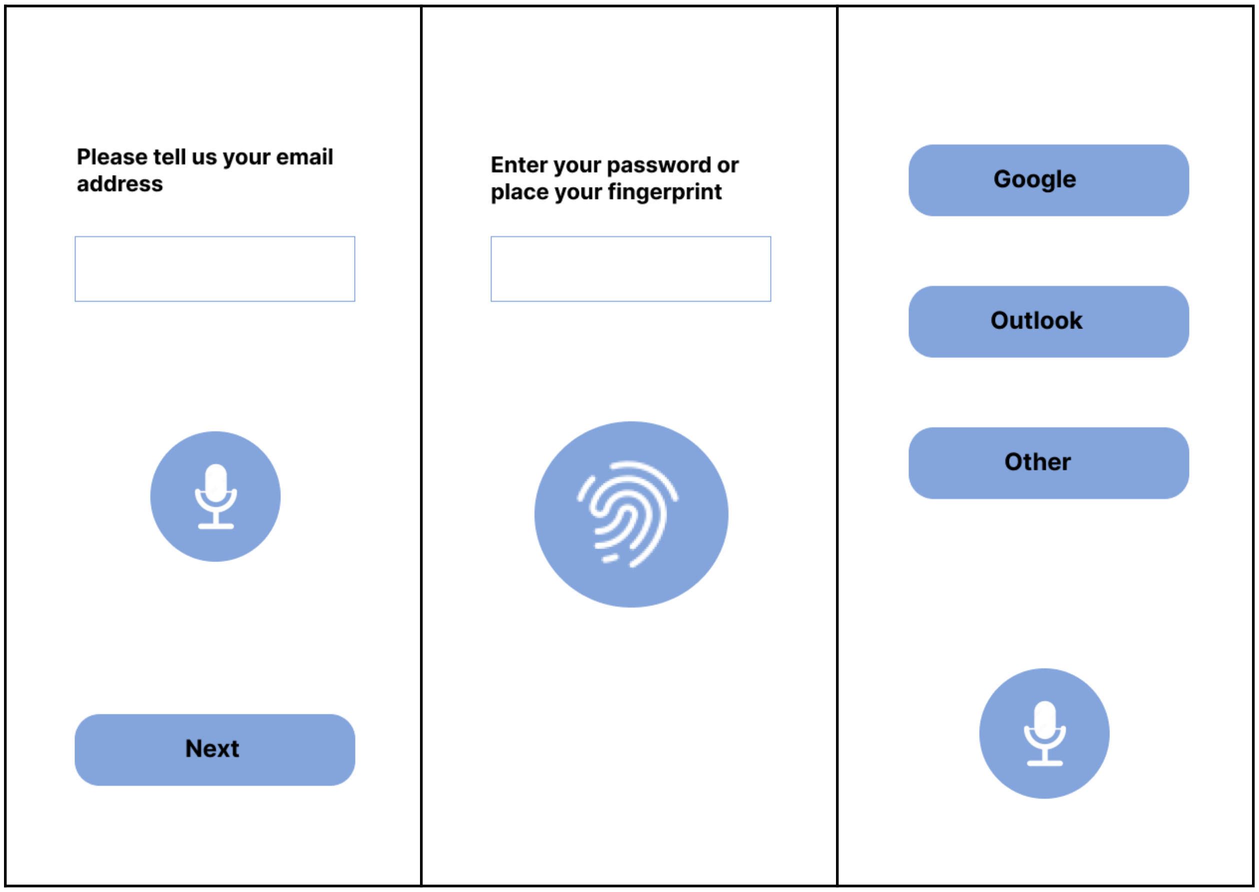
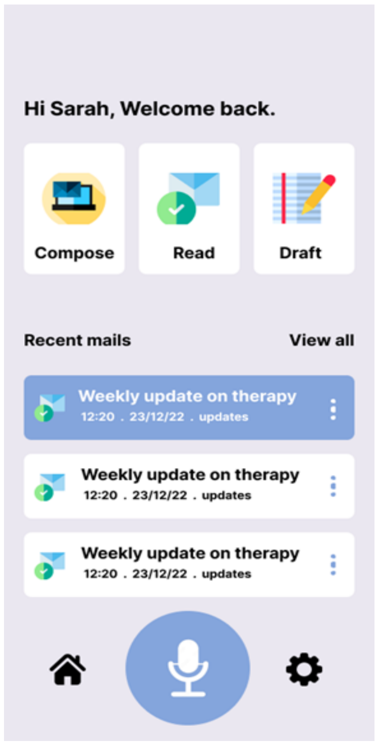

Please tell us your email address
Next
Enter your password or place your fingerprint
Google
Outlook
Other

Figure 3.0:Login Screen GUI

c. Home Screen

- The home screen includes three buttons, Compose, Read and Draft which is announced by a voice-over.
- The user can navigate using voice commands.
- A microphone icon in the bottom center can be used to provide voice input.
- To the right of the microphone icon is the settings icon. (Users can also select settings by saying settings to the input).


Hi Sarah, Welcome back.

Weekly update on therapy
12:20 . 23/12/22 . updates

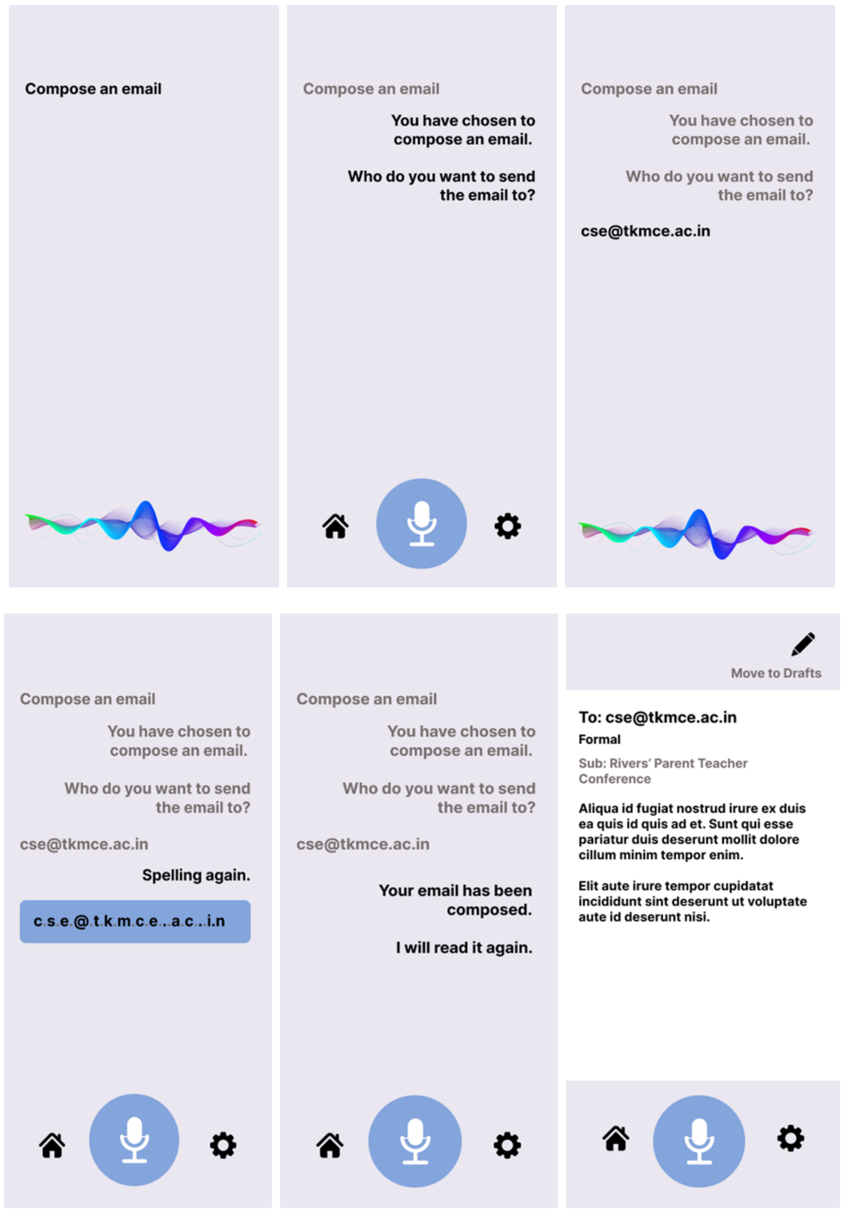
Weekly update on therapy
12:20 . 23/12/22 . updates

Weekly update on therapy
12:20 . 23/12/22 . updates

Figure 4.0: Home Screen GUI

d. Compose Screen

- The compose screen includes voice-over instructions on how to compose a new email.
- The user can input the recipient's email address, subject, and message using voice commands.
- To the right of the microphone icon is the settings icon and to the left is the home screen icon.



Compose an email

Compose an email

You have chosen to compose an email.

Who do you want to send the email to?

Compose an email

You have chosen to compose an email.

Who do you want to send the email to?

cse@tkmce.ac.in

Compose an email

You have chosen to compose an email.

Who do you want to send the email to?

cse@tkmce.ac.in

Spelling again.

c.s.e.@.t.k.m.c.e..a.c..i.n

Compose an email

You have chosen to compose an email.

Who do you want to send the email to?

cse@tkmce.ac.in

Your email has been composed.

I will read it again.

Move to Drafts

To: cse@tkmce.ac.in

Formal

Sub: Rivers' Parent Teacher Conference

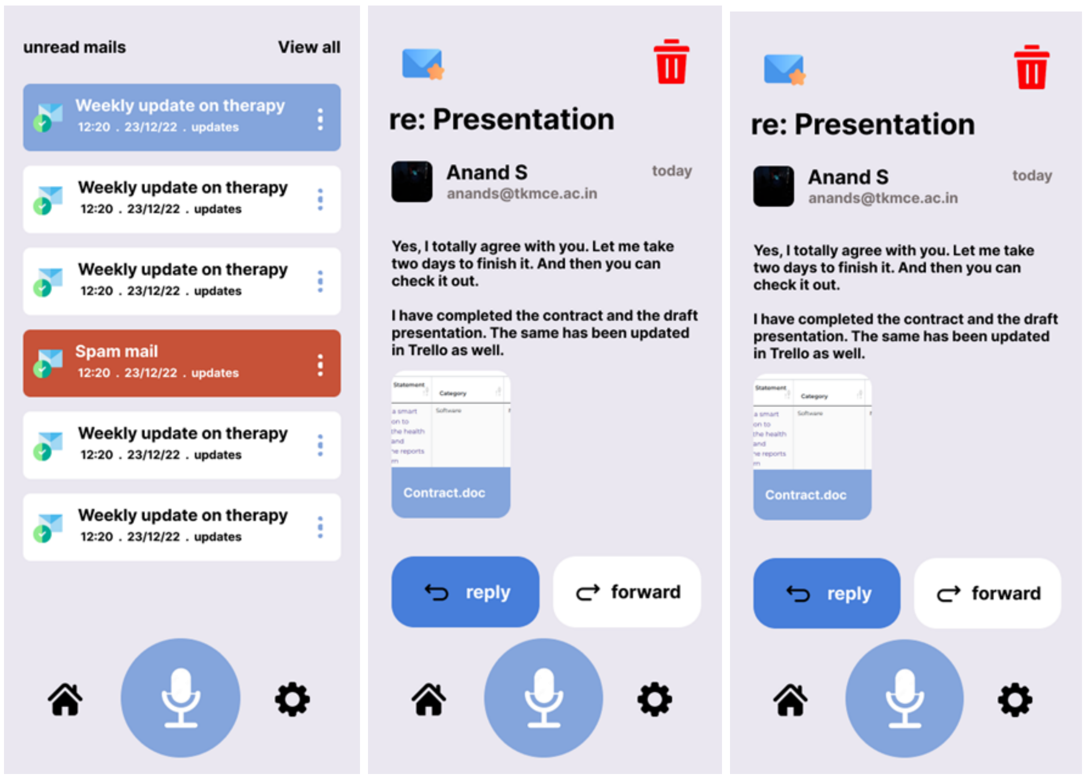
Aliqua id fugiat nostrud irure ex duis ea quis id quis ad et. Sunt qui esse pariatur duis deserunt mollit dolore cillum minim tempor enim.

Elit aute irure tempor cupidatat incididunt sint deserunt ut voluptate aute id deserunt nisi.

Figure 5.0: Compose Screen

e. Read Email Screen

- The read email screen includes a list of unread emails. The user can view all emails by clicking the view all button or saying it to the input.
- Unread emails are announced by voice-over. Email is selected as per the user input and the email's content is read out loud by a voice-over.
- There are voice commands and buttons to go back to the inbox, delete the email, reply, forward etc
- To the right of the microphone icon is the settings icon and to the left is the home screen icon.



Yes, I totally agree with you. Let me take two days to finish it. And then you can check it out.

I have completed the contract and the draft presentation. The same has been updated in Trello as well.

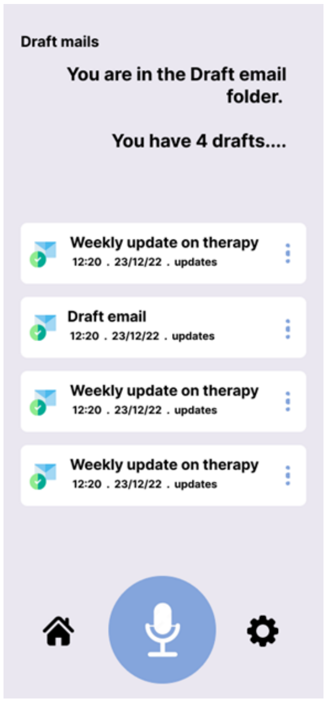
Yes, I totally agree with you. Let me take two days to finish it. And then you can check it out.

I have completed the contract and the draft presentation. The same has been updated in Trello as well.

Figure 6.0: Read Email Screen

f. Draft Screen

- The drafts are listed by subject and can be edited or sent using voice commands.
- To edit a draft email, the Compose screen is used, with To and subject prefilled.
- To the right of the microphone icon is the settings icon and to the left is the home screen icon.



Draft mails

You are in the Draft email folder.

You have 4 drafts....

Weekly update on therapy
12:20 . 23/12/22 . updates

Draft email
12:20 . 23/12/22 . updates

Weekly update on therapy
12:20 . 23/12/22 . updates

Weekly update on therapy
12:20 . 23/12/22 . updates

Figure 7.0: Draft Screen

g. Settings Screen

- Buttons for Account settings, Audio control, Sound control and support.
- As per the chosen option, corresponding fields are displayed.

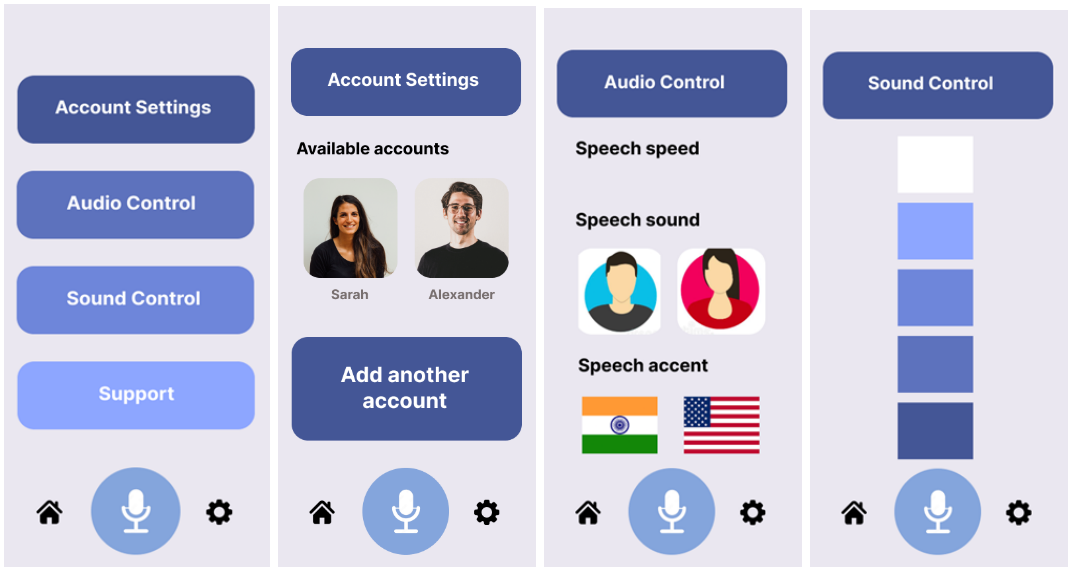


Figure 8.0: Settings Screen

4.2 Visual Design

- The GUI design will use high-contrast colors and larger font sizes for better visibility and readability.
- Icons and buttons will have descriptive labels in addition to their visual representation.
- Images will be accompanied by descriptive voice-over text where necessary.

4.3 Navigation Design

- The application will use a consistent navigation structure throughout
- Navigation will be designed with clear and descriptive labels to aid users with limited vision.

5. API Design

The Echo application utilizes two main APIs: Flutter and Django. Flutter provides built-in packages for text to speech and speech to text conversion, while Django is used as the back-end framework for the application.

5.1 Flutter API

Flutter provides a set of built-in APIs for text to speech and speech to text conversion. These APIs will be used to enable users to interact with the Echo application using their voice.

The Text to Speech API will convert text-based emails into audible speech, which can be listened to by the user. The Speech to Text API will convert the user's spoken commands into text format, which can then be used to navigate the application, compose new emails, and reply to existing ones.

5.2 Django API

Django is used as the back-end framework for the Echo application. The Django API is responsible for handling user requests and managing email data.

The Django API consists of several endpoints, including:

Authentication endpoint: Used to authenticate users and grant access to their email accounts.

Email retrieval endpoint: Used to retrieve email data from the email server.

Email composition endpoint: Used to compose and send new emails.

Email deletion endpoint: Used to delete emails from the user's inbox.

5.3 Natural Language Processing (NLP) API

The Echo application also plans to incorporate NLP functionality within Django to enhance the accuracy of speech to text conversion. NLP will be used to analyze the user's speech patterns, identify common phrases, and improve the overall accuracy of voice recognition within the application.

6. Technological Stacks

- Flutter framework:

  Flutter is an open-source UI framework for building high-performance, cross-platform mobile applications. It provides a fast development experience and supports a wide range of mobile platforms

- Dart programming language

  Dart is the primary programming language used in Flutter development. It is a modern, object-oriented language with features like garbage collection, asynchronous programming, and strong type checking.

- Speech recognition and synthesis libraries

  To enable voice-based interaction with the email application.We speech recognition and synthesis libraries within the flutter framework. Flutter provides plugins for both Google's Speech-to-Text API and Text-to-Speech API that can be used to implement these features.

- Email service provider API

  Email Service Provider (ESP) API is an interface provided by an email service provider such as gmail,outlook etc that allows developers to programmatically interact with the ESP's email sending and management capabilities.

- Dart Speech

  The Dart Speech package also provides a set of utilities for speech recognition and synthesis

7. Component Design

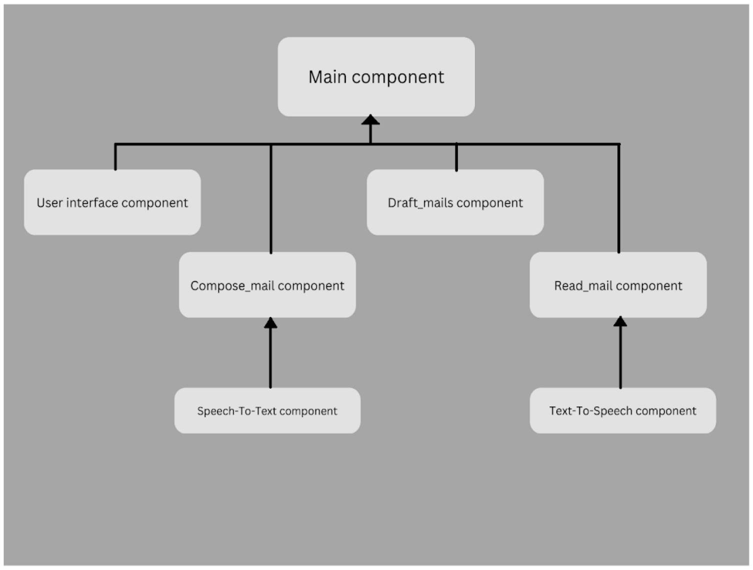


Figure 9.0: Component Design

- User interface component

  A user interface component is needed to provide feedback to the user and allow them to interact with the system even though primary input for the system is voice input.

- Compose_mail component

  This component is responsible for providing the user with a user-friendly interface for composing new email messages.It includes the features user Input ,confirmation and send.

  Speech-To-Text component is integrated with compose_mail component for the conversion of user voice inputs to text.

- Draft_mails component

  This component deals with draft mails.Users can review their unsend and incomplete compose mails and take actions.

- Read_mail component

  It is responsible for providing the user with a user-friendly interface for reading and managing their incoming email messages.

  Read_mail component should be able to convert the text of the email message into spoken words using a Text-To-Speech component.

  It also contains other features including filtering,mark as read/unread and delete.

8. Algorithm Design

Initialize the app:
a. Load the necessary libraries and modules for the app to function
b. Establish a connection to the server using Django framework
c. Authenticate the user and retrieve their email credentials

Main Loop:
a. Display the main menu options for the user:
- i. Compose a new email
- ii. Read incoming emails
- iii. Write Drafts
- iv. Navigate to a specific email
- v. Exit the app

b. Wait for user input, either through voice commands or GUI buttons

c. If user selects "ECHO COMPOSE":
- i. Prompt the user to speak the recipient's email address
- ii. Convert the speech input to text
- iii. Verify the email address is valid and prompt the user to confirm
- iv. Prompt the user to speak the email subject
- v. Convert the speech input to text
- vi. Prompt the user to speak the email body
- vii. Convert the speech input to text
- viii. Send the email

d. If user selects "ECHO READ":
- i. Retrieve the list of new emails from the email server
- ii. For each email, extract the subject, sender, and message body
- iii. convert the message body to speech
- iv. Display the email information to the user using GUI interface and also read the email information through voice

e. If user selects "ECHO LIST":
- i. Retrieve the list of sent emails from the email server
- ii. For each email, extract the subject, recipient, and message body

iii. Display the email information to the user using GUI interface and also read the email information through voice

f. If user selects "ECHO DRAFT":

i. Prompt the user to speak the email's subject
ii. Convert the speech input to text
iii. Prompt the user to speak the email's body
iv. Convert the speech input to text
v. Display the email information to the user using GUI interface and also read the email information through voice

g. If user selects "ECHO SPELL":

i. Display the email information letter by letter to the user using GUI interface and also read the spelling of email information through voice

h. If user selects "ECHO EXIT":

i. Terminate the main loop and exit the app

9. Data Design

- Data Storage
  - The application will need to store various types of data, including user account information and email messages. To do this, we will use a relational database management system (RDBMS) such as MySQL or PostgreSQL. The database will have the following tables:
  - Users: Stores user account information such as username and password.
  - Emails: Stores email messages along with metadata such as sender, recipient, subject, and timestamp.
  - We will use Django's built-in Object-Relational Mapping (ORM) framework to interact with the database. This will allow us to use Python code to define the structure of the database and interact with it, instead of writing raw SQL queries.

- Data Access
  - To access the data stored in the RDBMS, the Django ORM provides several APIs such as QuerySet, Model and Manager. This will allow us to access the data using object-oriented programming techniques.

- Data Transfer
  - The application will need to transfer data between the user's device and the email server. To do this, we will use the Simple Mail Transfer Protocol (SMTP) for sending email messages and the Internet Message Access Protocol (IMAP) for receiving email messages. We will use Python's built-in smtplib and imaplib modules to communicate with the email server.

- Data Backup and Recovery
  - To ensure that user data is not lost in the event of a system failure, we will implement regular backups of the database. We will also develop a recovery plan in case of a disaster, such as restoring the database from a backup or using a replication solution to maintain multiple copies of the data. We will use Django's built-in dumpdata and loaddata commands to backup and restore the database.

- Speech-to-Text Conversion
  - The application will use speech-to-text conversion to allow users to enter details, send and read emails. We will use a third-party API such as Google Cloud
  - Speech-to-Text or Amazon Transcribe to convert speech input to text.

- Speed Control
    - The application will allow users to control the speed of voice replies. This will be stored as a configuration setting for each user. However, since there are no other configuration settings, this will be stored in the Users table as a field.

10. Error Handling Design

- Network errors: In a voice-based email system, network errors such as slow internet connections can be common.The system handles these errors, with appropriate error messages and automatic retries when possible.Also when sending a mail in the absence of internet, the mail should be transferred to draft automatically.

- Error reporting: Whenever an error or crash occurs it will be reported to the server.

- Hardware error checking: Users can test the speakers and mic of their system to ensure they are fully operational.

- Clear and concise error messages: Since users will be interacting with the system using their voice, error messages should be clear, concise, and easy to understand.

- Person names,slangs etc handling: When we are inputting or reading such words the system cannot identify them.At this case the user can input or the system can read them character by character.

11. Performance Design

The performance of the Echo application is critical to ensure a smooth and efficient user experience. The following performance design considerations have been taken into account during the development of the application:

- Text to Speech and Speech to Text Conversion:
  The performance of the voice conversion feature is critical to ensure that the user can efficiently interact with the application. We will be using Flutter's built-in package for text to speech and speech to text conversion to ensure a fast and reliable conversion process. This package is known for its speed and accuracy, which will provide a seamless experience to the users. Additionally, we are also planning to implement NLP (Natural Language Processing) inside Django to further improve the accuracy of the conversion.

- Backend Framework:
  Django has been chosen as the backend framework for the Echo application. Django is a high-performance and scalable framework that can handle a large volume of requests. We will be optimizing the backend for better performance by using caching techniques and optimizing the database queries. The use of Django will ensure that the application can handle a high volume of requests without any performance issues.

- Frontend Framework:
  Flutter has been chosen as the frontend framework for the Echo application. Flutter is a high-performance framework that enables the creation of fast and efficient mobile applications. We will optimize the application's UI to ensure that it is lightweight and fast. This will ensure that the application is responsive and provides a smooth user experience.

- Server and Database:
  The performance of the server and database is critical to ensure that the application can handle a high volume of requests. We will be using a high-performance server and database to ensure that the application can handle a large volume of requests without any performance issues. We will also be optimizing the database queries to ensure that they are fast and efficient.

- Caching:
  Caching is a crucial technique to improve application performance. We will use caching to store frequently accessed data to reduce the number of database queries and improve application performance. This will ensure that the application is fast and responsive.

- Testing and Optimization:
  We will be conducting regular performance testing to ensure that the application is performing optimally. Any performance issues that are identified during testing will be addressed immediately to ensure a smooth and efficient user experience. We will also be optimizing the application to ensure that it is as fast and efficient as possible.

12. Security Design

- Secure authentication: It is ensured that only authorized users can access the voice-based email system. This might include password-based authentication or biometric authentication.
- Storage security: The system mainly uses the data from the database of the email provider.
- Compliance with data privacy regulations: The voice-based email system is subject to data privacy regulations such as GDPR.
- Scam protection: Using the email service provider features the software will let the user to ignore the scams and ad mails.

13. Testing Design

13.1 Testing Approach

The overall approach for testing the application will be a combination of manual and automated testing. Manual testing will be used to verify the user interface and user experience, while automated testing will be used to verify functionality, integration, and performance. Each feature of the application will be tested separately before testing the application as a whole.

13.2 Testing Scope

The scope of testing for this application will include the following types of testing:

1. Functional testing: To ensure that the application meets its functional requirements
2. Integration testing: To ensure that all components of the application work together seamlessly.
3. Performance testing: To ensure that the application performs within acceptable limits and responds to user input within 2 seconds and loads images and data within 5 seconds
4. Security testing: To verify that the application is secure and complies with GDPR and other privacy regulations
5. Usability testing: To test the application's ease of use and accessibility for blind and visually impaired users.

13.3 Test Cases

We will use the following test cases to verify that the application meets its requirements:

1. **Functional Testing**
    a. Test Case 1 - Email Account Setup
        Expected Result: Users should be able to set up their email accounts and log in to their Gmail or Outlook accounts and authenticate their credentials through voice commands.
    b. Test Case 2 - Voice Recognition and Email Formatting
        Expected Result: The application should be able to transcribe spoken text accurately, format the transcribed text into a readable email format, allow users to edit the transcribed text if necessary, and provide users with a confirmation option before sending the email.
    c. Test case 3 - Edit the transcribed text
        Expected Result: The user is able to edit the transcribed text before sending the email.
    d. Test case 4 - Confirming the email before sending

Expected Result: The user is able to confirm the email before sending it.

e. Test Case 5 - Voice Recognition Accuracy

Expected Result: The application should accurately transcribe spoken text, even in noisy environments or with accents or speech impediments. Tested by speaking with different accents, using different volumes or pitches, or speaking in noisy environments.

f. Test Case 6 - Email Attachment

Expected Result: The application should allow users to add attachments to their emails, either through voice commands or manual selection.

g. Test Case 7 - Email Forwarding

Expected Result: The application should allow users to forward received emails to other recipients using voice commands. Test this by receiving an email and forwarding it to a different email address or contact.

h. Test Case 8 - Email Reply

Expected Result: The application should allow users to reply to received emails using voice commands. Test this by receiving an email and replying to it with a voice command.

i. Test Case 9 - Email Deletion

Expected Result: The application should allow users to delete received or sent emails using voice commands. Test this by deleting received or sent emails using voice commands.

2. **Integration Testing**

a. Test Case 1 - Detect new incoming emails

Expected Result: The system extracts the text content of new emails as they arrive in the user's inbox.

b. Test Case 2 - Convert the email's text content into an audio format.

Expected Result: The system converts the email's text content into an audio format that is easily understandable by the user.

c. Test Case 3 - Control the speed and volume of the speech output

Expected Result: Users are able to control the speed and volume of the speech output through the system's user interface. Users are also able to pause, rewind, or skip the audio playback as needed through the system's user interface.

d. Test Case 4 - Audio output is clear and natural-sounding

Expected Result: The audio output is clear and natural-sounding and easily understandable by the user.

e. Test Case 5 - Handle multiple user accounts

Expected Result: The system can handle multiple user accounts and switch between them seamlessly, without requiring the user to log out and log back in.

3. **Performance Testing**
    a. Test Case 1 - Email Download Speed
    Expected Result: The application should download emails from the user's email provider in a timely manner.
    b. Test Case 2 - Response Time
    Expected Result: The application should respond to user input (e.g. voice commands) within acceptable limits and responds to user input within 2 seconds and load images and data within 5 seconds.
    c. Test Case 3 - Concurrent Users
    Expected Result: The application should be able to handle multiple users simultaneously without a significant decrease in performance.
    d. Test Case 4 - Load Testing
    Expected Result: The application should be able to handle a high volume of traffic without a significant decrease in performance.
    e. Test Case 5 - Stress Testing
    Expected Result: The application should be able to handle a sudden increase in traffic without crashing or experiencing significant performance degradation.
    f. Test Case 6 - Resource Utilization
    Expected Result: The application should use system resources efficiently and not consume an excessive amount of memory or CPU usage.

4. **Security testing**
    a. Test Case 1 - Authentication
    Expected Result: The application should require users to authenticate with a strong password or other secure authentication methods before accessing their email accounts.
    b. Test Case 3 - Encryption
    Expected Result: The application should encrypt user data in transit and at rest to protect against unauthorized access or interception.
    c. Test Case 5 - Session Management
    Expected Result: The application should manage user sessions securely to prevent unauthorized access or session hijacking. It is tested by attempting to access an active session from a different device or location.
    d. Test Case 6 - Error Handling
    Expected Result: The application should handle errors and exceptions securely to prevent information disclosure or other security vulnerabilities.
    e. Test Case 7 - Denial of Service (DoS) Attacks
    Expected Result: The application should be able to handle high volumes of traffic and prevent DoS attacks that could result in a loss of service.

f. Test Case 8 - Vulnerability Scanning
Expected Result: The application should undergo regular vulnerability scanning to identify and address potential security vulnerabilities.

5. **Usability testing**
   a. Test Case 1 - Navigation:
   Expected Result: Users can easily navigate through the application using voice commands and are able to access all the features and options without confusion or difficulty.
   b. Test Case 2 - Voice Recognition Accuracy:
   Expected Result: The application should accurately recognize and transcribe user voice commands, even in noisy environments or with accents or speech impediments.
   c. Test Case 3 - Feedback on User Actions:
   Expected Result: The application should provide clear and timely feedback to the user when an action is performed, such as confirming the user's command, indicating the status of a process, or providing error messages when necessary.
   d. Test Case 4 - Help and Documentation:
   Expected Result: The application should provide clear and easy-to-understand help and documentation that is accessible through voice commands or the user interface.
   e. Test Case 5 - Learnability:
   Expected Result: The application should be easy to learn and use for users with varying levels of experience and technical knowledge.
   f. Test Case 6 - Error Handling:
   Expected Result: The application should provide clear and helpful error messages when something goes wrong, and users should be able to easily recover from errors.
   g. Test Case 7 - Customization:
   Expected Result: Users should be able to customize the application's settings, such as language, speech rate, or email display format, to suit their preferences.
   h. Test Case 8 - Accessibility:
   Expected Result: The application should be accessible to users with disabilities, such as visual or motor impairments, and provide alternative ways to interact with the system.

13.4 Testing Tools

We will use the following testing tools to automate and facilitate testing:

A. Screen Reader: Use a screen reader tool such as NVDA (Non-Visual Desktop Access) to ensure that the application can be navigated and used by blind users who rely on screen readers.
B. Speech Recognition Software: Use speech recognition software such as Dragon Naturally Speaking to ensure that the application accurately transcribes voice commands and inputs.
C. Accessibility Testing Tools: Use accessibility testing tools such as a colour contrast analyzer, keyboard-only navigation testing tool, and screen magnification software to ensure that the application meets accessibility standards for users with low vision or motor disabilities.
D. Test Automation Tools: Use test automation tools such as Selenium and Appium to automate repetitive testing tasks such as login, email composition, and email reading.
E. Load Testing Tools: Use load testing tools such as JMeter or LoadRunner to simulate a large number of users accessing the application simultaneously and verify that the system can handle the load without crashing or slowing down.
F. Security Testing Tools: Use security testing tools such as OWASP, ZAP to perform penetration testing and identify potential security vulnerabilities in the application.
G. User Feedback Tools: Use user feedback tools such as Usabilla or UserTesting to gather feedback from blind users on the application's usability, accessibility, and overall user experience.

13.5 Test Environment

The test environment for Echo will include the following hardware, software, and network configurations:

A. Hardware: The app will be tested on a variety of devices, including smartphones and tablets running on iOS and Android operating systems.
B. Software: The app will be tested on the latest versions of iOS and Android operating systems.
C. Network: The app will be tested on both cellular and Wi-Fi networks with different signal strengths to ensure that it can function properly under different network conditions.

13.6 Test Data

A. Test email accounts:
   A set of test email accounts will be used to verify the functionality of the email setup feature. These email accounts will include both Gmail and Outlook accounts with varying authentication settings.
B. Sample emails:

A set of sample emails will be used to verify the email transcription, formatting, and sending features. These sample emails will include different types of content such as text, images, and attachments, and will be used to test the accuracy of the email transcription feature as well as the ability to edit and send emails using voice commands.

C. Noise data:

A set of noise data will be used to test the accuracy of the voice recognition feature in noisy environments. This data will include recordings of different types of noise such as traffic, background conversations, and music, and will be used to simulate the types of environments in which the app might be used.

D. Accent data:

A set of accent data will be used to test the accuracy of the voice recognition feature for different accents. This data will include recordings of people with different accents, including but not limited to American, British, Australian, and Indian accents.

E. Speech impediment data:

A set of speech impediment data will be used to test the accuracy of the voice recognition feature for people with speech impediments. This data will include recordings of people with different types of speech impediments, including but not limited to stuttering, lisps, and voice tremors.

13.7 Acceptance Criteria

The following acceptance criteria must be met for the application to be considered ready for release:

- All test cases have been executed and passed
- Performance benchmarks have been met, including page load times and response times
- Security requirements have been met, including data encryption and user authentication
- Usability metrics have been met, including user satisfaction and ease of use.
- Compatibility with different operating systems and devices
- Compatibility with different email providers (e.g., Gmail, Outlook, Yahoo)
- Compliance with accessibility guidelines
- Compliance with industry standards and regulations
- Integration with other assistive technologies

13. Maintenance Design

This section outlines the plan for maintaining the voice-based email application for visually impaired users. The purpose of this section is to provide a framework for the team responsible for maintenance to ensure the software remains up-to-date, secure, and reliable.

Maintenance Plan

The voice-based email application will undergo regular maintenance updates to ensure its continued functionality and usability. The maintenance team will be responsible for implementing these updates, which will occur on a quarterly basis. In the event of urgent updates, emergency maintenance may be performed as needed.

The following strategies will be implemented to ensure effective maintenance:

1. Regular updates: The application will be updated regularly to ensure that it remains compatible with the latest screen reader software and email providers. This will also include fixing any bugs that may arise and addressing user feedback.
2. Backward compatibility: The application will be designed to maintain backward compatibility with previous versions of the application to ensure that users who have not upgraded to the latest version can continue to use the application without interruption.
3. User feedback: A mechanism will be put in place to gather user feedback, which will be used to inform future updates and improvements to the application.
4. Data backup and recovery: Regular backups of user data will be performed to ensure that user data is not lost in the event of a system failure. A recovery plan will also be in place to restore user data in the event of a catastrophic failure.
5. Security updates: The application will be monitored regularly for security vulnerabilities, and updates will be made as necessary to ensure that the application remains secure and protects user data.
6. User training: As the application is updated and new features are added, user training materials will be updated to ensure that users are aware of these changes and know how to use the application effectively.

Change Management

Changes to the voice-based email application will be managed through a formal process that includes requesting changes, reviewing changes, and implementing changes. Change requests will be submitted through a designated ticketing system, which will be reviewed by the development team. Changes will be implemented following a thorough review and approval process to ensure the integrity of the software.

Version Control

The voice-based email application will use GitHub to track changes to the software system. The team responsible for maintenance will utilize this system to manage code changes, track revisions, and deploy updates. Any changes made to the software will be tracked, reviewed, and tested before being deployed to ensure the stability and reliability of the system.

Documentation

The voice-based email application will maintain up-to-date documentation, including user manuals, technical documentation, and release notes. The team responsible for maintenance will ensure that all documentation is accurate and up-to-date, providing end-users with the information they need to effectively use the software system.

14. Deployment Design

The mobile application will be deployed to the Google Play Store and Apple App Store, which will allow users to download and install the app on their Android and iOS devices. The deployment process involves publishing the app to the app stores and ensuring that it meets their respective requirements and guidelines.

To ensure optimal performance and user experience, the app will be regularly updated with bug fixes and new features. These updates will be pushed to the app stores, and users will be notified of the availability of the updates on their devices. The app will also include a built-in feedback mechanism to allow users to report bugs and suggest new features.

In terms of security, the app will employ industry-standard security practices to protect user data and prevent unauthorized access. This will include secure storage and transmission of user data, as well as strict access control measures for the app's backend services.

To ensure scalability and availability, the Echo app will be designed to work with a scalable cloud infrastructure. This will allow the app to handle increasing numbers of users and requests without compromising performance or availability. Additionally, the app's cloud infrastructure will be monitored regularly to ensure that it is operating optimally.

Finally, the deployment design will include a maintenance plan to ensure that the app continues to function as intended over time. This will involve ongoing monitoring of the app's performance and user feedback, as well as regular updates to ensure that the app remains compatible with new versions of the operating systems and other dependencies.

Appendix A: Record of Changes

Version	**Date**	**Author**	**Description of change**
1.0	30/03/2023	Salihu Ahamed	Added System and Application Architecture
1.0	30/03/2023	Ashwin Binu	Added Data and Algorithm Design
1.0	31/03/2023	Fathima Nooha	Started GUI Design and Testing Design
1.0	01/04/2023	Deepak P	Edited Component and Error Handling Design
1.0	01/04/2023	Fathima Nooha	Updated Maintenance Design
1.0	01/04/2023	Ashwin Binu	Edited Performance Design part
1.0	02/04/2023	Deepak P	Added Security Design
1.0	02/04/2023	Salihu Ahamed	Finalized document for submission

Appendix B: Acronyms

Acronym	Literal translation
API	Application Programming Interface
GUI	Graphical User Interface
NLP	Natural Language Processing
UI	User Interface
ESP	Email Service Provider
RDBMS	Relational Database Management System
ORM	Object-Relational Mapping
SQL	Structured Query Language
SMTP	Simple Mail Transfer Protocol
IMAP	Internet Message Access Protocol
GDPR	General Data Protection Regulation

Appendix C: Referenced Documents

Document Name	Document URL	Issuance Date
"Building Accessible Android Apps" by Google Developers	https://developer.android.com/guide/topics/ui/accessibility/principles	September 2021
"Best Practices for Developing Accessible Mobile Applications" by the World Wide Web Consortium (W3C)	https://www.w3.org/WAI/standards-guidelines/mobile/	December 2022
"Voice User Interface Design" by the Google Developers	https://developers.google.com/assistant/conversation-design	January 2023