

**Technical Design Document**

**(TDD)**

MailSpeak Application (MSA)

University of Maryland Global Campus (UMGC)

Software Engineering (SWEN) 670

Fall Cohort 2022

Team B

November 5, 2022

**Document Control**

**Document Information**

|  | **Information** |
| --- | --- |
| Document Identification | USPS- MSA-TDD-20220917-Fall2022 |
| Document Owner | UMGC SWEN 670 |
| Issue Date | September 17, 2022 |
| Last Saved Date | November 5, 2022 |
| File Name | TDD – Team B - MSA.docx |

**Document History**

| Version | Issue Date | Changes |
| --- | --- | --- |
| 0.1 | 9/14/2022 | Initial Draft |
| 0.2 | 9/17/2022 | Review Updates |
| 1.0 | 9/17/2022 | Final Deliverable |
| 1.1 | 10/9/2022 | Logo and name change updates |
| 2.0 | 10/29/2022 | Post-development Update |
| 2.1 | 10/30/2022 | References style and document suite updates |
| 2.2 | 11/4/2022 | Requirements matrix updates |
| 3.0 | 11/5/2022 | Final deliverable |

Approval Signatures

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ☐ APPROVE  ☐ DISAPPROVE | |  | | --- | |  | | Tatiana Kozhevnikova, Product Owner  Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ | |  | |
| ☐ APPROVE  ☐ DISAPPROVE | |  | | --- | |  | | Michael Conatser, Project Manager  Date: \_\_\_\_\_\_\_\_\_\_\_\_\_ | |

**Table of Contents**

[1 Introduction 5](#_Toc118469293)

[1.1 Purpose 5](#_Toc118469294)

[1.2 Scope 5](#_Toc118469295)

[1.3 Overview 5](#_Toc118469296)

[1.4 Project Documentation 6](#_Toc118469297)

[1.4.1 Project Suite of Documents 6](#_Toc118469298)

[1.4.2 Document References 6](#_Toc118469299)

[2 System Overview 6](#_Toc118469300)

[3 System Architecture 7](#_Toc118469301)

[3.1 Architectural Design 7](#_Toc118469302)

[3.2 Decomposition Description 8](#_Toc118469303)

[3.2.1 Mail Search Decomposition 8](#_Toc118469304)

[3.2.2 Chatbot Decomposition 9](#_Toc118469305)

[3.3 Exception Handling 9](#_Toc118469306)

[3.3.1 Mail Search Exceptions 9](#_Toc118469307)

[3.3.2 Chatbot Exceptions 9](#_Toc118469308)

[3.4 Design Rationale 9](#_Toc118469309)

[4 Data Design 10](#_Toc118469310)

[4.1 Data Description 10](#_Toc118469311)

[4.2 Data Dictionary 12](#_Toc118469312)

[4.2.1 ChatbotService 12](#_Toc118469313)

[4.2.2 MailEntity 12](#_Toc118469314)

[4.2.3 MailService 12](#_Toc118469315)

[5 Component Design 13](#_Toc118469316)

[5.1 ChatbotService 13](#_Toc118469317)

[5.2 MailService 13](#_Toc118469318)

[5.3 Client – Email Login 14](#_Toc118469319)

[5.4 Semantics (Flutter Plugin) 15](#_Toc118469320)

[5.5 Gestures (Flutter Plugin) 15](#_Toc118469321)

[6 Human Interface Design 15](#_Toc118469322)

[6.1 Overview of User Interface 15](#_Toc118469323)

[6.2 Screen Images 15](#_Toc118469324)

[6.2.1 Application Login Screen 16](#_Toc118469325)

[6.2.2 Mail Search Screen 17](#_Toc118469326)

[6.2.3 Chatbot Screen 18](#_Toc118469327)

[6.2.4 Application Settings Screen 19](#_Toc118469328)

[7 Requirements Matrix 20](#_Toc118469329)

[Attachment A – Acronyms and Definitions 22](#_Toc118469330)

# Introduction

## Purpose

The Technical Design Document (TDD) will detail the implementation of requirements for the Mail Speak Application (MSA) as defined in Software Requirements Specification (SRS). This document describes the system architecture, data design, components design and user interface design of MSA.

The intended audience of this document includes Software Engineers and Testers, Technical Writers, Project Manager and other project stakeholders involved in the development process of the MSA.

## Scope

The scope of the project is to enhance the pre-existing functionality of the Mail Speak Application and implement additional features which will broaden app accessibility and usability for visually impaired, as well as sighted users. The baseline functionality of the MSA provides access to users’ emails through the login into their email accounts with implemented accessibility features such as text-to-speech and speech-to-text. The full scope of the project is divided between two teams, A and B.

**Scope of the project for Team B includes:**

* + Enhance Accessibility features by providing Gesture and Voice-driven commands, and improving Read Screen functionality across all features of MSA.
  + Improve Search View, to allow search for specific date range/sender/keywords, store and utilize object character recognition (OCR) content of mail pieces to enhance the search, and provide feedback to USPS on conducted searches.
  + Implement Chatbot functionality that provides access to all features of MSA.

**Outside of Team B’s scope:**

* + Enhancements to Mail View.
  + Implementation of Notification View.
  + Providing Internal Feedback on cyclic consumer behavior.
  + Integration of the Voice Assistant such as Google Assistant and Siri.

## Overview

This TDD consists of the following sections:

* + **System Overview -** provides a general description of MSA’s new features and enhancements to major functionality and design.
  + **System Architecture** - lays down a top-level design for incorporated features and enhancements of MSA. Also, a basis for more detailed design is provided such as a description of the subsystem components, their relationships and dependencies, and how the exceptions will be handled.
  + **Data Design** - describes how data will be used by the system and lists all data entities and descriptions used by the system.
  + **Components Design -** describes the functionality of each new or enhanced component with detailed pseudo code for correct implementation.
  + **Human Interface Design** - provides User Interface (UI) design mockups and descriptions that support the construction of application screens.
  + **Requirements Matrix -** represents all functional requirements from SRS and components that satisfy them.

## Project Documentation

### Project Suite of Documents

This Technical Design Document is part of a set of essential documents created to adequately manage, control and deliver the MailSpeak Application. Artifacts that are provided within the document package contain vital information for the application’s ongoing support and operation throughout its life cycle. Each document is created within the specific Milestone of the project. Therefore, the version and date of some documents could be marked as “to be determined” (TBD) in Table 1.1.

Table 1‑1 - Project Documentation Package

| Document | Version | Date |
| --- | --- | --- |
| Project Management Plan (PMP) | 4.0 | 11/05/2022 |
| Software Requirements Specification | 4.0 | 11/05/2022 |
| Technical Design Document | 3.0 | 11/05/2022 |
| Software Test Plan | 3.0 | 11/05/2022 |
| Programmer Guide | 2.0 | 11/05/2022 |
| Development and Operations Guide | 2.0 | 11/05/2022 |
| User Guide | 1.0 | 11/05/2022 |
| Test Report | 1.0 | 11/05/2022 |

### Document References

During the process of writing current TDD the following documents were referenced:

* Team B. (2022). *United Global Master Coders Team B PMP*. <https://umgcdev361.sharepoint.com/:w:/r/sites/SWEN670Fall2022/Shared%20Documents/Team%20B%20Channel/Milestone%201%20(SAT%20SEP%203)/ProjectManager-Project-Plan-Template.docx?d=w671384dfe89d46d7a2583b60416fb909&csf=1&web=1&e=xeRN2o>
* Team B. (2022). *United Global Master Coders Team B SRS*. <https://umgcdev361.sharepoint.com/:w:/r/sites/SWEN670Fall2022/Shared%20Documents/Team%20B%20Channel/Milestone%201%20(SAT%20SEP%203)/Informed%20Delivery%20Enhancement%20Team%20B%20SRS.docx?d=w9b27ad03c5c145a09edc6b22427bc8ba&csf=1&web=1&e=85qCsH>
* MozillaWiki (2022, August 9) *Thunderbird: Autoconfiguration: ConfigFileFormat.* MozillaWiki. <https://wiki.mozilla.org/Thunderbird:Autoconfiguration:ConfigFileFormat>
* University of Maryland Global Campus (UMGC). (2022) *SWEN 670 capstone project management system.* <https://umgc-cappms.azurewebsites.net/download/48433bae-ed2e-4593-9285-80715ad69039----Team-A-SoftwareRequirementSpecification.docx>

# System Overview

This project extends and improves the functionality of the existing Mail Speak Application. Additional UI components will be implemented to support enhancements. The Search View will allow users to perform a search for a specific date range, search using keywords and sender information. The keywords field will display suggestions for typed character sequences from the local mail repository. Google Cloud Vision will be used to process mail images and populate the local mail repository. Search results will be presented to the user as a list of emails that satisfy search criteria.

The new Chatbot feature will be integrated into the existing MSA. The Chatbot UI component will be persistent across the application’s screens. The interaction with the Chatbot will be in the form of messenger, which will provide screen-related commands suggestions and process commands specified by the user.

Gesture and Voice command functionality will be modified to utilize the native accessibility features of the Operating System (OS). The Flutter Semantics widget is needed to annotate the widget tree with the description of the meaning of the widgets. This information will be used by native OS accessibility tools. Gestures Flutter plugin will be utilized to customize gestures.

# System Architecture

The System enhancements being made by Team B focus on improving Mail Search, Gesture Commands, and Enabling a Chatbot for communication. While the Gesture Commands simply use a single Application Programming Interface (API) to provide access to the system level functionality, the Search and Chatbot will require a more carefully architected approach. This approach will be outlined in the following sections.

## Architectural Design

The basic architecture of the MSA can be broken down into 5 main layers.

* UI Layer: How the user interacts with the system
* Service Layer: Application specific services that take inputs and perform domain specific functions to return the result to the UI
* External UI Components: Third Party tools to enable system level functions on various devices
* Database Layer: Storage system of the application
* External APIs: Third Party tools for performing complicated functions such as Image Processing and Mail fetching

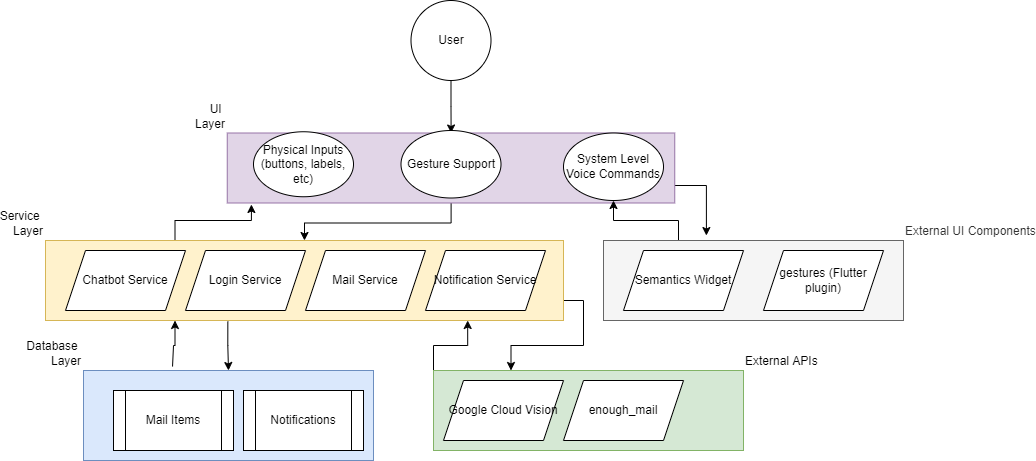


Figure 3.1 - System Architecture Diagram

## Decomposition Description

Rather than detailing each piece of functionality in the MSA, separate diagrams will be provided for the individual components being enhanced/implemented by Team B.

### Mail Search Decomposition

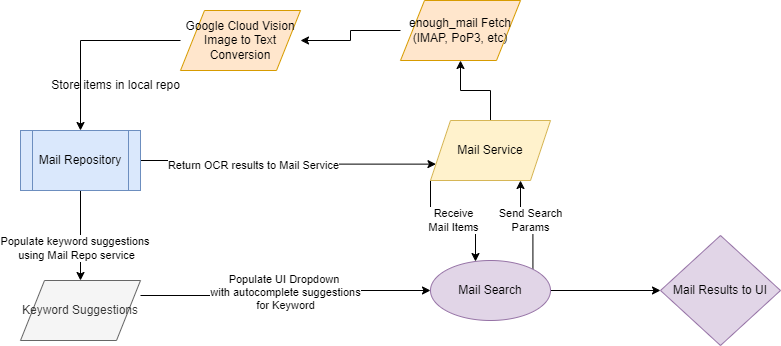


Figure 3.2.1 - Mail Search Decomposition

The updated Mail Search fetches items using an Internet Message Access Protocol (IMAP) connection, but now passes search parameters that will be used to filter the data returned. In addition to filtering, the images will be processed and converted to text that will be stored in a local mail repository for suggestions to populate the keyword mail search filter. Upon successful search, the results are returned to the UI.

### Chatbot Decomposition

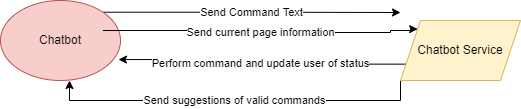


Figure 3.3 - Chatbot Decomposition

The Chatbot user interface will provide the user a direct way to input commands that will be interpreted by the Chatbot service based on the current page the user is on of the application. The Chatbot service will also make suggestions to inform the user of the commands they can perform based on their current page as well. The Chatbot user interface is wrapped in an External UI to provide a clean, familiar messenger display to distinguish messages from the application from those of the user.

## Exception Handling

Below is a brief summary of the exceptions that will be handled by the new features of the application. If a user message is to be displayed that will be specified; otherwise, the exception will be logged for the development team to investigate.

### Mail Search Exceptions

* **FetchMailException(String username):** This exception will be thrown when the user information fails to fetch using the IMAP connection. A detailed message of the error can be displayed if needed.
* **ImageTranslationException(int mailIdentifier):** This exception notes that an image has failed to be translated into text information. The identifier can be passed for debugging purposes or logging to USPS.
* **MailStoreException:** This exception is thrown when the system fails to write to the Mail Repository. A message will be displayed to the user to notify if the issue was due to size constraints or permissions.

### Chatbot Exceptions

* **InvalidCommandException(string command):** The command could not be recognized by the Chatbot Service. The system will show this error as a response by the chatbot will state the command was not understood.
* **UnableToPerformCommandException(string command):** This error will be displayed in the chat window when a command that should be able to be performed fails. The Chatbot will display a meaningful error to the user as to what occurred and any potential resolution possible.

## Design Rationale

The overall architecture chosen was decided to organize and enhance an existing application to allow for more simplicity on iterations. With the inclusion of multiple third-party technologies, it was important to separate the UI technologies from the External API technologies so it is easy to determine what layer of the application will have access to the technologies.

When deciding upon having a database layer within the application versus streaming the data live, it was apparent that repeated streaming and image processing using the Google Cloud Vision API could get very expensive for the customer. This led to the architectural decision of storing the OCR results in a local database along with an Identification (ID) to map to the emails that would be fetched.

With the decision of storing email information locally, building a service to interact with that data become obvious to extract the domain logic from the actual data layer. With the addition of the Mail Service to perform this function, we can also leverage the service to make keyword suggestions when performing a search based on the stored data we have parsed. This will allow a much more informed user experience as they will be able to see valid keywords before even returning the full results of their search query.

Not leveraging a third party chatbot tool such as DialogFlow was a conscious decision to avoid locking the customer into too many paid services. Providing a predefined set of system functions along with suggesting commands to the user should allow for a natural interaction with the chatbot without requiring the heavy processing needed for natural language processing and machine learning. Separating this into a Chatbot service will allow future development efforts to easily replace this system with a more robust system if desired.

# Data Design

## Data Description

The core functionalities being worked as part of this project are the Mail Search and Chatbot system, with included integration for gesture and voice command throughout the application. While the accessibility libraries will be mostly handled by third-party tools and libraries, a Mail Service and Chatbot Service will need to be created to handle the domain specific interaction as it is much more complex.

The Mail Service will be a dedicated class with access to the Google Cloud Vision API for translating images to text, IMAP library for accessing the user’s emails, and a mail repository for storing the translated data. The mail repository will be inside the application and store the entities fetched and translated so they can be easily retrieved as needed. This repository will also make it much faster for the service to perform search queries on mail items for the new enhanced mail search functionality. The format of this data will be JavaScript Object Notation (JSON) to store multiple mail items.

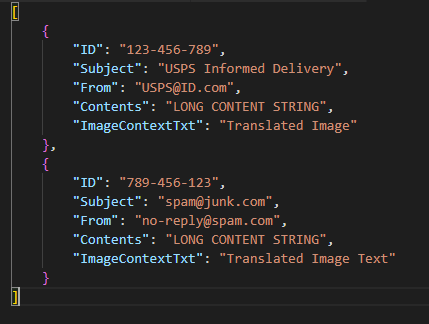


Figure 4.1 Example of JSON Formatted Mail Data

The Chatbot Service will be another dedicated class that will store a Dictionary of pages within the application and the list of functions that can be performed on each page. These functions will be stored in string format and used to easily trigger the desired responses once submitted to the Chatbot. There will also be the ability to pass parameters with the commands for more complex functions to be run (Example: From the Mail Search page, the user could enter “Search boss” and the chatbot could perform a search query using the keyword “boss” and return the list of results to the user). A pseudocode example of how the enumerated type of SiteAreas would work with a HashMap and Chatbot Service to provide this functionality is provided in Figurer 4.2 below.

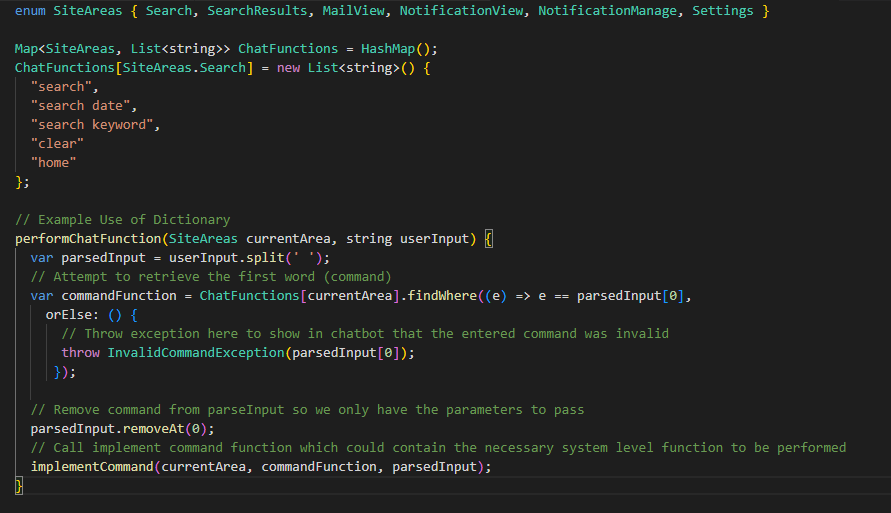


Figure 4.2 Example Storage and Use of Chatbot Service Commands

## Data Dictionary

Alphabetically list the system entities or major data along with their types and descriptions. For OO design list the objects and its attributes, methods and method parameters

The ChatbotService class will be responsible for handling user input and converting it to system functions based on the current page and text entered by the user. This service will also be able to display suggestions of commands for the user to perform based on the current page.

### ChatbotService

*Attributes:* ChatFunctions

*Methods:* performChatFunction, getChatSuggestions, sendChatResponse

### MailEntity

The MailEntity will be responsible for storing the mail item metadata. This class will get converted to and from JSON format as items are requested and stored.

*Attributes:* ID, Subject, From, Contents, ImageContentsTxt

*Methods*: N/A

### MailService

The MailService class will be responsible for fetching mail items from storage and filtering the items given search parameters. This service will also be able to update the stored items using the IMAP connection.

*Attributes:* mailData

*Methods:* fetchMail, saveMail, refreshFromIMAP, getKeywordSuggestions, convertImageToText

# Component Design

In this section, we take a closer look at what each component does in a more systematic way. Below is the list of primary classes being added/updated in the system along with their detailed description and capabilities. Pseudocode will be provided to explain more complex functions as needed.

## ChatbotService

Class Name: ChatbotService

Class Description/Purpose: Providing suggestions and interpreting user input in relation to a set of predefined commands. These commands are then translated to actual system functions passing any provided parameters.

Class Modifiers:

* public
* async

Class Inheritance:

* N/A

Class Attributes:

* **ChatFunctions**: HashMap containing enumerated type of system area as the key along with a list of commands that can be performed.

Exceptions Thrown:

* **InvalidCommandException:** Valid command could not be parsed
* **UnableToPerformCommandException:** Valid command failed to be performed

Class Constructors:

* **ChatbotService():** No needed parameters. Would populate ChatFunctions dictionary on creation.

Class Methods:

* **getChatSuggestions(SiteAreas currentPage):** Retrieve available commands for user input based on the current page.
* **performChatFunction(SiteAreas currentPage, string userInput):** Parses and performs user specified command.
* **sendChatResponse(String output):** Updates the UI with a specified chat response based on the performed function.

## MailService

Class Name: MailService

Class Description/Purpose: Interacting with the stored Mail data (in JSON format). This class will convert that data to a proper object and allow fetching and updating of that data as needed.

Class Modifiers:

* public
* async

Class Inheritance:

* N/A

Class Attributes:

* **mailData**: file location/json string of data stored

Exceptions Thrown:

* **FetchMailException:** When mail fails to be retrieved from IMAP
* **ImageTranslationException:** When image cannot be translated by Google Cloud Vision
* **MailStoreException:** When the system fails to write to the JSON file for storage

Class Constructors:

* **MailService():** No needed parameters. Would load mailData file from hardcoded location

Class Methods:

* **convertImageToText(byte[] data):** Passes image format to Google Cloud Vision for processing and returns the string contents.
* **fetchMail(string? Keyword, DateTime? startDate, DateTime? endDate**): Retrieve mail with optional parameters to filter the results
* **getKeywordSuggestions(string? Keyword):** Looks up potential keyword suggestions filtering by user entered param. Results are returned in ListView format for display.
* **refreshFromIMAP:** Calls to refresh data from the IMAP.
* **saveMail():** Calls to refresh data from the IMAP, convert images to text as needed, and update data in the JSON file.

## Client – Email Login

Class Name: Client

Class Description/Purpose: To establish a connection with an email service. Email protocols to support include Post Office Protocol version 3 (POP3), and IMAP. Configurations of email servers are provided by xml lookups and/or Mozilla’s ongoing support with Auto Configuration xml (Thunderbird: Autoconfigruation:ConfigFileFormat – MozillaWiki, n.d)

Class Modifier:

* public
* async

Class Inheritance:

* N/A

Class Attributes:

* N/A

Exceptions Thrown:

* **MailException:** General API message from enough\_mail package
* **PopException/ImapException**: Connection with email server failed via specified email protocol

Class Constructors:

* **Client():** this should be a static class.

Class Methods:

* **getImapClient(String username, String password, String token = “”)**: attempts to connect to email using IMAP protocol; if token is provided, try to connect via Open Authorization (OAuth) protocol.
  + Internally utilize oAuth2Login and oAuth2Bearer functions from enough\_mail package.
* **getPOPClient(String username, String password)**: attempts to connect to email using POP3 protocol
* **getServerConfigFromEmail(String email)**: looks up connection settings for email server based on given email

## Semantics (Flutter Plugin)

The Semantics Flutter plugin will be used to add components in the UI layer that can integrate directly with native components on the respective device for voice and reading functionality. Using this prevents the team from having to write separate code to handle voice commands and readability on multiple devices as Semantics will seamlessly integrate with the device’s accessibility features.

## Gestures (Flutter Plugin)

The Gestures Flutter plugin will be used to add components in the UI layer that can integrate directly with native components on the respective device for gesture commands. Using this prevents the team from having to write separate code to handle gestures on multiple devices the Gestures plugin will seamlessly integrate with the device’s accessibility features.

# Human Interface Design

## Overview of User Interface

The application UI will be developed using the Flutter development framework and the Dart programming language. Google's Cloud Platform (GCP) services will be extensively used to detect user inputs and provide options on information that becomes available.

The application will inform the user when a daily digest email arrives and listen for keywords/phrases to interact with them. It will then convert those phrases to text to determine what actions to trigger on the application. The user will also have the ability to press the mic button to request details, scan mail to aid the user verbally and inform them of what mail they received. The application can also scan physical mail and verbally tell the user the sender and the recipient information displayed. It will also check for logos from daily digest emails and scan mail to provide additional information on the email’s sender.

## Screen Images

The following mock-up designs, in the form of screenshots, illustrate the main aspects of the application’s UI. Each screenshot is accompanied by an explanation of the user’s expected experience when interacting with the respective aspects. These mock-up designs consist of the following screenshots:

* + Application Login Screen
  + Main Menu Screen
  + Mail Search Screen
  + Chatbot Screen
  + Application Settings Screen

### Application Login Screen

This is the first user interface screen that the user shall see upon launching the app. The screen shall display two text fields for an email address and password respectively so that the user can input the login credentials for their email address/account that has been registered with United States Postal Service (USPS) Informed Delivery. The screen also shall include separate links beneath the email address and password fields that the user can tap to register for Informed Delivery or view the MSA “Terms and Conditions” and “Privacy Policy”. If the user taps the “Retrieve Mail” button after inputting their login credentials (and assuming the app has verified the login credentials as valid), the app shall display the Main Menu screen as seen in section 6.2.2.

Graphical user interface, application, website

Description automatically generated

Figure 6.2.1 - MSA Login Screen

1. Main Menu Screen

When the user has successfully logged into the application, they shall see the following screen that displays the main features that are available to the user in the form of buttons. These features include: Search Mail, Daily Digest, Scan Mail, Upload Email, Notifications, and Chatbot. At the top of the screen the menu icon is located on the right, that opens the dropdown menu with Settings and Logout options. At the bottom of the screen, a quick navigation bar shall be visible and accessible to the user regardless of which screen they are on and lets the user access the Main Menu (represented with a home icon), navigate to Search Mail screen(represented with an envelope icon), Scan Mail (represented with a camera icon), view their Notifications (represented with a bell icon), or interact with Chatbot (represented by a little robot icon).

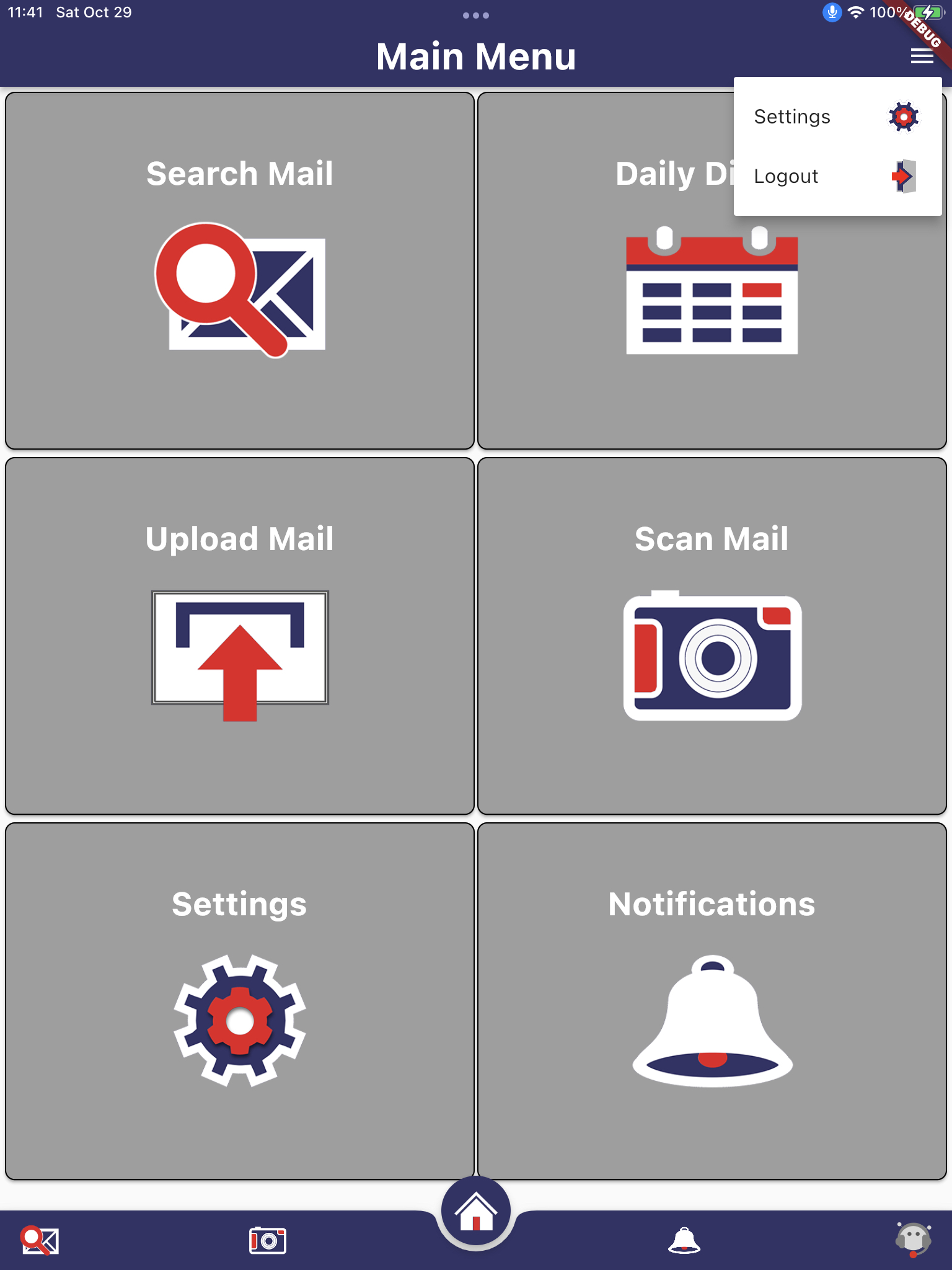


Figure 6.2.2 - MSA Main Menu Screen

### Mail Search Screen

If the user chooses to access the Search Mail feature, the following screen shall be displayed. The user shall determine a time period when searching. They can either select a date from the calendar view or use the start and end date range selectors to enter a valid date range. The keyword field shall have the capability of predictive text to aid the user. As the user enters a value, the system shall automatically provide logical assumed values and provide those for the user to select or allow the user to continue entering letters until the autocomplete comes up with the most logically relevant word.

Graphical user interface, application, Teams, calendar

Description automatically generated

Figure 6.2.3 - MSA Mail Search View

### Chatbot Screen

If the user chooses to access the Chatbot feature, the following screen shall be displayed. Initially, the user shall be prompted with a set of opening questions by the Chatbot that shall allow the system to begin narrowing down the issue or task the user is inquiring about. Once the system has sufficient information based on user responses, it shall be able to respond to the user’s inquiry. The Chatbot messages shall be represented with grey text bubbles and the user’s messages shall be represented with purple text bubbles.

Graphical user interface, text, application, Teams

Description automatically generated

Figure 6.2.4 - MSA Chatbot View

### Application Settings Screen

In the following user interface, the user shall be able to control the settings used by the application. Settings shall display “Terms and Conditions” and “Privacy Policy” links, as well as an ability to delete all local data.

Graphical user interface, text, application

Description automatically generated

Figure 6.2.5 - MSA Settings Screen

# Requirements Matrix

Provides a cross-reference that traces components and data structures to the requirements in the SRS document. The table below shows which system components satisfy each of the Use Case requirements from the SRS. For each requirement the corresponding component is specified within this document that would satisfy that requirement.

Table 7.1 – Requirements Traceability Matrix

| System Req. Number | System Ref. Item | Component Identifier | Component Item |
| --- | --- | --- | --- |
| 3.1.1 | Searches inbox for mail pieces that were sent within a given date range and/or contain a given keyword | 5.2 | MailService |
| 3.1.2.1 | Chatbot provides the user with suggested commands | 5.1 | ChatbotService |
| 3.1.2.2 | Chatbot provides the user with access to help documentation | 5.1 | ChatbotService |
| 3.1.2.3 | Chatbot allows the user to add new notifications | 5.1 | ChatbotService |
| 3.1.2.4 | Chatbot allows the user to remove notifications | 5.1 | ChatbotService |
| 3.1.2.5 | Chatbot allows the user to search for mail sent by a given sender | 5.1 | ChatbotService |
| 3.1.2.6 | Chatbot allows the user to search for mail sent on a given date | 5.1 | ChatbotService |
| 3.1.2.7 | Chatbot allows the user to search for mail sent within a given date range | 5.1 | ChatbotService |
| 3.1.3.1 | Navigates application | 5.5 | Gestures (Flutter Plugin) |
| 3.1.3.2 | Navigates application | 5.5 | Gestures (Flutter Plugin) |
| 3.1.3.3 | Navigates application | 5.5 | Gestures (Flutter Plugin) |
| 3.1.3.4 | Dismisses subscreen | 5.5 | Gestures (Flutter Plugin) |
| 3.1.4.1 | Logs in to application | 5.3  5.4 | Client – Email Login  Semantics (Flutter Plugin) |
| 3.1.4.2 | Shows top bar menu | 5.4 | Semantics (Flutter Plugin) |
| 3.1.4.3 | Shows settings | 5.4 | Semantics (Flutter Plugin) |
| 3.1.4.4 | Logs out | 5.4 | Semantics (Flutter Plugin) |
| 3.1.4.5 | Opens mail search | 5.4 | Semantics (Flutter Plugin) |
| 3.1.4.6 | Searches inbox for mail pieces that were sent within a given date range and/or contain a given keyword | 5.2  5.4 | MailService  Semantics (Flutter Plugin) |
| 3.1.4.7 | Opens individual mail piece | 5.2  5.4 | MailService  Semantics (Flutter Plugin) |
| 3.1.4.8 | Opens notifications | 5.4 | Semantics (Flutter Plugin) |
| 3.1.4.9 | Adds new notifications | 5.4 | Semantics (Flutter Plugin) |
| 3.1.4.10 | Removes notifications | 5.4 | Semantics (Flutter Plugin) |
| 3.1.4.11 | Opens chatbot | 5.4 | Semantics (Flutter Plugin) |
| 3.1.4.12 | Executes commands with chatbot | 5.1  5.4 | ChatbotService  Semantics (Flutter Plugin) |
| 3.1.4.13 | Opens settings | 5.4 | Semantics (Flutter Plugin) |
| 3.1.4.14 | Displays “Terms and Conditions” | 5.4 | Semantics (Flutter Plugin) |
| 3.1.4.15 | Displays “Privacy Policy” | 5.4 | Semantics (Flutter Plugin) |
| 3.1.4.16 | Dismisses dialog | 5.4 | Semantics (Flutter Plugin) |
| 3.1.4.17 | Opens daily digest | 5.2  5.4 | MailService  Semantics (Flutter Plugin) |
| 3.1.4.18 | Displays next mail piece in daily digest | 5.2  5.4 | MailService  Semantics (Flutter Plugin) |
| 3.1.4.19 | Displays previous mail piece in daily digest | 5.2  5.4 | MailService  Semantics (Flutter Plugin) |
| 3.1.4.20 | Opens upload mail | 5.4 | Semantics (Flutter Plugin) |
| 3.1.4.21 | Opens scan mail | 5.4 | Semantics (Flutter Plugin) |
| 3.1.4.22 | Opens main menu | 5.4 | Semantics (Flutter Plugin) |
| 3.1.4.23 | Navigates application | 5.4 | Semantics (Flutter Plugin) |
| 3.1.5.1 | Reads all elements on the screen | 5.4 | Semantics (Flutter Plugin) |
| 3.1.5.2 | Reads single element on the screen | 5.4 | Semantics (Flutter Plugin) |

Attachment A – Acronyms and Definitions

|  |  |
| --- | --- |
| Term | Meaning |
| API | Application Programming Interface |
| GCP | Google Cloud Platform |
| ID | Identification |
| IMAP | Internet Message Access Protocol |
| JSON | JavaScript Object Notation |
| MSA | Mail Speak Application |
| OAuth | Open Authorization protocol |
| OCR | Object Character Recognition |
| OS | Operating System |
| PMP | Project Management Plan |
| POP3 | Post Office Protocol version 3 |
| SRS | Software Requirements Specification |
| SWEN | Software Engineering |
| TBD | To Be Determined |
| TDD | Technical Design Document |
| UI | User Interface |
| UMGC | University of Maryland Global Campus |
| USPS | United States Postal Service |