A picture containing logo

Description automatically generated

Technical Design Document

University of Maryland Global Campus

SWEN 670 – Team A

Fall Semester

Version 1.1

October 29, 2022

Document Control

Document Information

|  |  |
| --- | --- |
| © | Information |
| Document Id | USPS-IDE-TDD |
| Document Owner | UMGC SWEN 670 TEAM A |
| Issue Date | September 17, 2022 |
| Last Saved Date | October 29, 2022 |
| File Name | USPS\_Informed\_Delivery\_App\_Enhancements\_TDD\_Team\_A.docx |

Document History

|  |  |  |
| --- | --- | --- |
| Version | Issue Date | Changes |
| 1.0 | 09/17/2022 | Initial Document |
| 1.2 | 10/29/2022 | Post-development updates |
|  |  |  |
|  |  |  |

Contents

[1 Introduction 1](#_Toc118703394)

[1.1 Purpose 1](#_Toc118703395)

[1.2 Project Documents 1](#_Toc118703396)

[1.3 Overview 1](#_Toc118703397)

[1.4 Scope 2](#_Toc118703398)

[1.4.1 In Scope 2](#_Toc118703399)

[1.4.2 Out of Scope 3](#_Toc118703400)

[2 System Overview 3](#_Toc118703401)

[3 Architectural Design 4](#_Toc118703402)

[3.1 Decomposition Description 5](#_Toc118703403)

[4 Data Design 6](#_Toc118703404)

[4.1 Application Data Handling 6](#_Toc118703405)

[4.2 Third Party Data Components 7](#_Toc118703406)

[4.2.1 JSON Request 7](#_Toc118703407)

[4.2.2 JSON Response 7](#_Toc118703408)

[4.3 Application Data Model 8](#_Toc118703409)

[4.4 Application Database Schema 9](#_Toc118703410)

[5 Component Design 9](#_Toc118703411)

[5.1 User Interface (Views) Component 10](#_Toc118703412)

[5.2 Management / Service Components 10](#_Toc118703413)

[5.3 Model Components 11](#_Toc118703414)

[5.3.1 Application Feedback Component 12](#_Toc118703415)

[5.3.2 Image–to–Text Component 12](#_Toc118703416)

[5.3.3 QR–Barcode Component 12](#_Toc118703417)

[5.3.4 Text-to-Link Component 13](#_Toc118703418)

[5.3.5 Voice Assistant Component 13](#_Toc118703419)

[5.4 Component Diagram 15](#_Toc118703420)

[6 Human Interface Design 16](#_Toc118703421)

[6.1 User Flow 16](#_Toc118703422)

[6.2 Screen Images 17](#_Toc118703423)

[6.2.1 Mail login credentials screen 17](#_Toc118703424)

[6.2.2 Main user interface screen 20](#_Toc118703425)

[6.2.3 Application settings screen 21](#_Toc118703426)

[6.2.4 Search interface screen 22](#_Toc118703427)

[6.2.5 Search results screen 23](#_Toc118703428)

[6.2.6 Individual Mailpiece display screen 24](#_Toc118703429)

[6.2.7 In-App notifications screen 25](#_Toc118703430)

[6.2.8 Manage notifications screen 26](#_Toc118703431)

[6.2.9 Chatbot interface screen 27](#_Toc118703432)

[7 Requirement Matrix 28](#_Toc118703433)

[8 Appendices 30](#_Toc118703434)

[1.1 Definitions, Acronyms, Abbreviations 30](#_Toc118703435)

[8.1 References 30](#_Toc118703436)

# Introduction

The United States Postal Service (USPS) is an agency that transports mail to and from residential customers. To better assist their customers, USPS offers a free subscription service called Informed Delivery, allowing users to receive images of their incoming mail seven days a week. A mobile application has been developed to assist visually impaired users accessing the USPS Informed Delivery email service. The product owner has requested additional features that will allow customers to consume and interact with the application better. The application is intended to run on mobile devices and will be a third party application called “Mailspeak”.

## Purpose

The purpose of this Technical Design Document is to describe the design and architecture of enhancements that are to be added to the USPS Informed Delivery Application. The sections of this document will outline the application’s scope, architectural design, data design, component design, and user interface design.

This document is intended to be used by Software Developers/Testers and Project Managers as a reference when implementing the design and for conveying the design to the USPS technology team.

## Project Documents

This Technical Design Document is part of a set of documents created to aid in developing the USPS Informed Delivery Application and to provide artifacts with vital information for the application’s ongoing support and operation throughout its life cycle.

**The following documents are included in the entire documentation package:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Document | Version | Date |
| 1 | Project Management Plan (PMP) | 1.2 | 9-17-2022 |
| 2 | Software Requirements Specification (SRS) | 1.2 | 10-29-2022 |
| 3 | Technical Design Document (TDD) | 1.2 | 10-29-2022 |
| 4 | Software Test Plan (STP) | 1.1 | 10-29-2022 |
| 5 | Programmers Guide (PG) | 1.1 | 11-05-2022 |
| 6 | Deployment and Operations (DevOps) | 1.1 | 11-05-2022 |
| 7 | User Guide (UG) | 1.0 | 11-05-2022 |
| 8 | Test Report (TR) | 1.0 | 11-05-2022 |

## Overview

The purpose of the USPS Informed Delivery App is to better enhance the user experience of the USPS Informed Delivery Service. For the application to interact with the user and their associated Informed Delivery content, the application will require the user to create a login account linked to their Informed Delivery subscription. The subscription utilizes the user’s indicated email address to deliver information for the user’s Informed Delivery digest. The application currently can take images from the Informed Delivery email, retrieve associated text on the image, and then have the information read to the user. Additionally, the user can scan Quick Response (QR) codes.

The product owner wants to add numerous enhancements to the existing application. Therefore, team A will be responsible for implementing the following requirements, included in four main categories.

* The first category includes items associated with email. For example, the application will add the following functionality:
  + Open an email from a search term.
  + Add the ability to call, email, or send a text message to the mail sender within a particular email.
  + Add relevant hyperlinks associated with each email and mail item.
* The second category includes adding a mobile device notification to the user about incoming mail from a specified sender or keyword. The notifications will also be available within the application. Tapping on the system notification or in-app notification will take the user to the associated email message.
* The third category includes tracking application behavior and making this data available to the product owner. Also, provide tracking on the links and barcodes visited.
* The fourth category includes integrating the voice assistant with Google Assistant.

Refer to the Software Requirements Specification for additional detail on each feature.

## Scope

The scope of this document is to provide a technical description of the design and architecture of the USPS Informed Delivery App for the intended readers. This document is broken down into sections that will detail the aspects of the application. The in-scope areas that this document will address are the following:

### In Scope

* **Architectural Design**: This section will detail the design of the application in regard to what devices are intended to be used and what third-party web-based functionality is intended to be provided.
* **Data Design**: This section will detail how data is utilized by the application. This includes where it is coming from and being sent to, where it is being stored, and what type of data the system will utilize for the app to function.
* **Component Design**: This section will describe the components of the application that make the application provide the intended features, including components implemented within the application and third-party resources utilized.
* **Human Interface Design**: This section will detail the functionality of the human interface design of the Mailspeak Application and how it is intended to interact with the user.

### Out of Scope

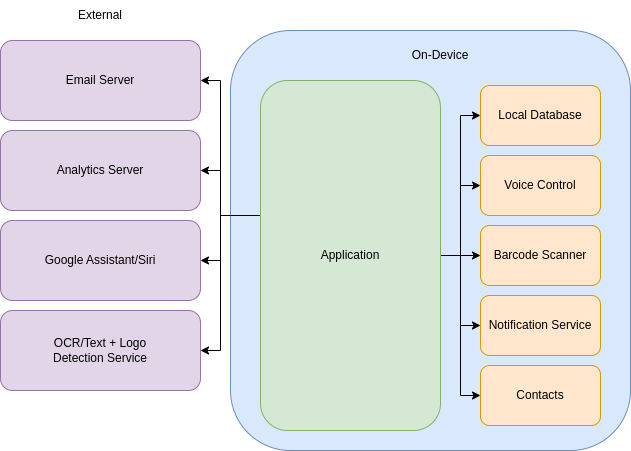
* **Accessibility**: All accessibility features, including voice input and screen reading, are not within the scope of Team A’s requirements. All accessibility features will be covered by Team B’s documentation.
* **Push Notification (OneSignal)** – The addition of a push notification service may be an enhancement that would benefit the application.

# System Overview

The main goal is to enhance the Mailspeak Application with a set of features that builds on the success of the last development initiative. To continue these customer-focused improvements, the product owner has requested an additional set of enhancements to provide even greater control over how a user may interact with the USPS Informed Delivery mail digest message and other email messages. The improvements in this scope will focus on search functionality, notifications and alerts, improved content recognition, chatbot support, integration with digital assistants, and several other features.

The application is intended to be installed on Apple mobile devices running iOS, or smartphones running the Android OS. These two platforms account for almost all smartphone users. To be available to install on these devices, the application will be made available for download for free within the Apple App Store and Google Play app store. It assumes the user has a smartphone with wireless or cellular data to send and receive data. It assumes the user already has an existing web-based email account and has a physical address where they receive USPS mail. The user can sign up for the Informed Delivery service through the application, but this is a link to the process hosted on the web by USPS.

There are two classes of interactions that this application will implement, on-device and external. Each feature may interact with zero or many of these services. The overall interactions with these on-device and external systems are as follows:

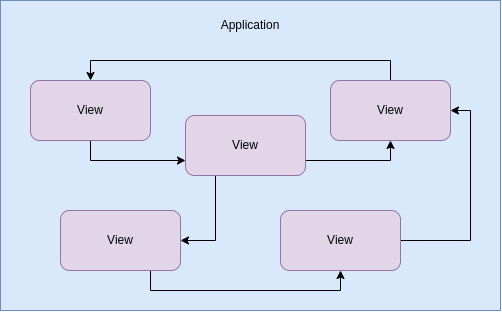


The application layer will communicate with these services to accomplish the various goals of each feature, but all servers/services outside of the application layer will be considered external to the application and will fall outside the scope of development. Instead, the application will interact with those services via clients or utility libraries provided by Flutter or Dart.

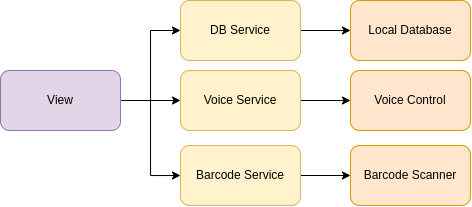
# Architectural Design

The architectural design depicts the backend components of the Mailspeak Application. It also shows the flow of data, from retrieving the data to the output of the returned data. The backend components include image processing to detect handwritten text, logos, and typed text, QR/ barcode processing components, and address validation using USPS Web Tools API. Google Cloud Platform is being used to encompass most of the architecture behind the Mailspeak application.

This application is a graphicly based user interface, which means we will architect the software around the concept of “views”. The current implementation, which this document and application are extending, uses this pattern. That means that each page of the application will encapsulate the logic required for that specific interface and will provide links to other views.



As described in the System Overview section of this document, this application will interact with a number of services to provide the functionality for both the user-facing and backend features. In order to allow for easy testing and exchanging of services in the future, our teams will employ an [“onion architecture”](https://medium.com/@shivendraodean/software-architecture-the-onion-architecture-1b235bec1dec), where the use of interfaces will define contracts between the application layer of the software and the services being used. This interfacing layer will provide a means for shifting to different services for the same use case, if need be, for example, changing from a local database to a cloud-based one.



## Decomposition Description

When the user opens the application, there are steps that need to occur for the app to perform its functionalities. The functionalities are the various components within the app. For the purpose of this design document, the components of the backend will be highlighted.

# Data Design

Data for the application will be transmitted over the internet, stored on the user’s mobile device, and sent to web services. The user can perform some app functions without an internet connection, but for full functionality, the application requires an internet connection.

## Application Data Handling

The application, when operating, will search for USPS informed delivery email data from the customer's existing email account using the username and password provided by the user for authentication. The images will be further processed through the web-based Google Vision application programming interface (API) to provide relevant text associated with the image. In addition, barcodes will be processed within the application using a barcode reader. Email text data will be stored along with associated text data associated with images and barcodes in a local database within the application. Any data to be stored in the local mobile device shall be encrypted.

The product owner would like to have anonymous application usage data. Google Analytics will be utilized; a free plugin is available for the application that tracks usage statistics through frequent data sent to the Google Analytics server that is collected within the application.

Additionally, the application will be able to access the user’s phone contacts list for generating calls, emails, or text messages while within the application. Finally, the application will be able to share data with the user’s phone voice assistant, Siri for Apple, and Assistant for Google. The exact format of this data will be documented in the software development phase.

User settings will be stored locally on the device disk using key-value pairs with methods included in the Flutter Software Development Kit (SDK).

Data shall be transmitted and received using JavaScript Object Notation (JSON). JSON is a simple format for storing and transmitting key-value data pairs. Flutter has integral functions for converting JSON to application objects.

JSON has simple syntax rules. Data is in key-value pairs, and separated by commas. Curly braces define objects, and square brackets represent arrays. An example of the mail piece object storing link data is below.

Mail Piece { [

"id": 1040503950,

"email\_id": 2022-10-08 10402940954,

"sender": “Home Depot”,

"image\_text": “Shop at Home Depot”,

"timestamp": 2022-10-08-XXXX,

"scanImgCID": 104950-34o30549480,

"uspsMID": 305986904059,

"image\_bytes": dfk49slg40slti49,

"featured\_html": XXX

] };

## Third Party Data Components

For the Google Cloud Vision API, JSON objects invoke requests and receive responses. These objects consist of name and value pairs hierarchically. For basic transmission, it will consist of three names. The first is the parent’s name, either a request sent to Cloud Vision API to process the image or a response from Cloud Vision to the application or user. These two JSON files will be used to obtain and receive the required information to create our application’s friendly JSON for transmission.

### JSON Request

The request will have an image and features name, which are required fields for Cloud Vision to process. In the image, it will send under content the base64 encoded version of the processing image. The device is responsible for converting the encoded string results to content. The Features name will consist of a type name, which will have the feature type used as its value. For this project scope, we will focus on TEXT\_DETECTION and LOGO\_DETECTION. These features can also be performed concurrently by adding them as an additional type under features.

### JSON Response

For the TEXT\_DETECTION feature, the response JSON will consist of textAnnotation and fullTextAnnotation objects, respectively. The first shown is the textAnnotation with all of the text identified as one followed by each word found in its description value. The next object is the fullTextAnnotation object which shows all the text in pages and is refined to blocks, paragraphs, words, and letters.

For Logo Detection, Google Vision API provides similar results as Text\_Detection. However, instead of having a TextAnnotation object, it consists of logoAnnotation objects. The logoAnnotation object consists of attributes, including mid, description, score, and boundingPoly. For this project, we will only focus on the description attribute.

## Application Data Model

Due to the complexity of the Vision JSON objects, especially when figuring out the sender and recipient information, a new JSON object will be created with the information needed for the frontend to invoke Text to Speech. These objects are based on models. It contains all required information found on the features performed. The JSON object will have the following format.

{ “outputs”: [ {

mailObject: [ { “type”: “value”

“name”: “value”,

“address”: “value”,

“validated”: “value”

},

],

LogoObject: { “description”: [ “name”, “value” ] }

codeObject: { “type:: “value”

“link” : “value”

}

}

]

}

Objects are created and added to the JSON when serialized if data is available, parsed, and retrieved from the cloud vision API response. This model intends to make it easier for User Interface to get the information required for data presentation and accurate voice dialog. See table 1 for a description of each attribute.

**Table 1:**

JSON object attributes

| **Object** | **Attribute** | **Description** |
| --- | --- | --- |
| mailObject | type | “sender”, “recipient” or “unknown” |
|  | name | company or person’s name |
|  | address | Name’s address to include city, state, and zip |
|  | validation | If the address was confirmed to exist using other apis |
| logoObject | Description | Word describing logo |
| codeObject | Type | Determines whether image found was a “QR” or “Bar” Code |
|  | Link | Points the user to where to find more information |

## Application Database Schema

Once the data has been processed, the mail pieces derived from the JSON objects defined in Section 4.3 are then persisted to a local SQLite database to allow for fast and efficient caching of processed mail pieces. In addition, notification subscriptions are also stored here, allowing for the generation of notifications based on simple SQL queries as new subscriptions and mail pieces are added to the database. The following table schema is used for this application:

|  |
| --- |
| TABLE mail\_piece:  id STRING UNIQUE NOT NULL  email\_id STRING  sender STRING  image\_text STRING  timestamp INTEGER  Links STRING?  Emails STRING?  Phones STRING?    TABLE notification\_subscription:  keyword STRING UNIQUE NOT NULL    TABLE notification:  mail\_piece\_id STRING  subscription\_keyword STRING NOT NULL |
|  |

# Component Design

The application will be organized into components by program features to achieve a manageable level of separation of concerns. The application will consist of three major component groups. User interface components will organize how the application interacts with the user and screen. Management / service components handle the primary functions of the application, and model components define objects. The application model components will also consist of the following third-party components or plugins: Image-to-Text and QR-Barcode Scanning, QR-Barcode Component, and USPS Address Validation. Pertinent data extracted from the mail piece image, such as sender information and affiliated links, is returned in a JSON format.

Every Flutter/Dart application must start with a main.dart file with a void main( ) function which contains information to start the application. Additional files for UI, management, and models will be provided with relevant file and class names to help with component organization.

To avoid creating an unresponsive UI, network operations such as email processing will be performed using asynchronous operations.

## User Interface (Views) Component

The User Interface will include graphical screens designed for a touchscreen mobile device to allow for user input and display of data for each process. This is the primary component the user will interact with. This will also include a voice interface and will rely on the mobile device speaker and microphone for audio input and output.

|  |  |
| --- | --- |
| **View** | **Description** |
| User Sign-In | Authenticates user account (email + password) |
| Main Menu (Home) | Main page for routing the user to application functions |
| Digest View | Handles displaying the data from the most recent Informed Delivery Daily Digest email the user selects during Email Search. |
| Mail Piece Search | Handles obtaining user input used by the model to query the local database for specific stored mail pieces |
| Mail Piece Display | Handles displaying retrieved emails, including displaying links and providing sender contact information |
| Display Links and Image OCR Text | Handles interface for displaying and redirecting the user to associated mail piece links |
| Upload Mail piece | Allows user to manually input their own mail piece |
| Scan | Allows user to access scanning functions such as QR code readers for mail pieces |
| Notifications | Notifies user when a specified mail piece from a sender arrives |
| Google Assistant | Handles interface between the application and mobile device voice assistant. |
| Application Settings | Handles obtaining user input for user-selectable application options |

Table 1 User Views and Descriptions

Application data and functions will be handled by backend classes and processes, allowing the user interface to be reconstructed as needed when application states change.

## Management / Service Components

The managers will be a set of classes responsible for managing the logic flow between the user interface and data model components. The primary components to handle and control are emails, notifications, and general settings, monitoring, and requests.

| **Model** | **Description** |
| --- | --- |
| General Manager | Handles logic for application settings, voice assistant, and application tracking |
| Mail Fetcher | Handles creating mail pieces from USPS informed delivery emails, executing OCR scan processes, and calling for required email gathering functions. |
| Mail Utility | Handles obtaining emails from the email server. |
| Mail Piece Service | Handles gathering mail pieces based on search queries provided from the Search UI page. |
| SQ Lite Database | Handles creating, storing, and retrieving database components for mail pieces. |
| Mail Notifier | Handles settings related to mail piece notifications. |
| Mail Loader | Handles retrieving mail piece from the local database. |
| Chatbot Service | Handles chatbot services. |
| Application Settings | Handles settings for general application settings. |
| Cache Service | Automatically executes at startup of application to obtain the latest informed delivery emails and storage of mail pieces. |
| Mail Piece Storage | Assists with storing mail piece data to the SQ Lite Database service. |
| Email Manager | Handles interface between email UI pages and email retrieval, storage, and link generation. |
| Analytics Services | Handles communication with Google Analytics server for application feedback. |
| Assistant Service | Handles logic for voice activated features with the Android OS. |

Table 2 Management / Services and Description

## Model Components

The model components handle the logic for implementing the features of the application.

| **Model** | **Description** |
| --- | --- |
| Address | Defines a postal address. |
| Application Function | Utilized by the Voice Assistant functionality to translate “intents” into application feature responses. |
|  |  |
| Code | Bar/QR code object created for Google Cloud Vision processing of barcodes and QR codes, providing access to the interpreted link. |
| Digest | Defines the MIME message, and associated attachments of an Informed Delivery email |
| Logo | Logo object created for Google Cloud Vision processing of logos, providing access to the text name. |
| Mail Piece | Defines a mail piece, including id, image id, timestamp received |
| Mail Response | Overall parent object utilized to return data from the Google Cloud Vision processing functions. |
| Mail Search Parameters | Handles building a search query for mail pieces |
| Search Criteria | Query parameters for conducting searches of mail pieces, including keywords, start, and end dates. |
| Notification | Defines an individual notification instance at a specific time and for a specific piece of mail. |
| Notification Subscription | Holds data for keywords for wwhich a user wants to receive notifications. |

Table 3 Model Components and Descriptions

### Application Feedback Component

The Application Feedback Component handles all internal and external logging required by the customer. This component utilizes standard flutter logging to report errors and generic logging at differing levels of granularity commonly used in the industry. External logging will transmit user metrics to a Google Firebase analytics server. Data for events such as clicking on links will be transmitted to the analytics server for easy user digestion through the analytics dashboard.

### Image–to–Text Component

The Image-to-Text Component takes an image of the mail piece from the email as input, using Google Cloud Vision API to recognize text and logos. The Vision API utilizes optical character recognition (OCR) to recognize text from images. For logos, the API utilizes the Logo Detection feature to recognize popular product logos. The USPS Address API validates and standardizes addresses detected by the OCR API, to minimize uncertainties that may occur with character detection.

### QR–Barcode Component

The QR-Barcode Scanning Component recognizes QR codes, barcodes, and links on the given mail piece image. The Google Machine Learning (ML) Kit Barcode Scanning plugin supports barcode detection and scanning. The API supports linear and 2D barcode formats, regardless of the image orientation. To ensure greater accuracy of the scan, Google Cloud Vision API utilizes Object Localization to determine the relative locations of barcodes on the mail piece image and generates the coordinates for a bounding polygon. For instance, a QR code is identified by the Object Localization feature of the API as a “2D barcode” and the coordinates for the vertices of its bounding polygon are returned in a JSON format. The bounded image is then passed in as input for the barcode scanner object.  The barcode scanner will then scan the image and will determine the type of barcode. The barcode data will be returned with the barcode type if the image is a valid barcode. If the image is a QR code, the data returned will be the imbedded URL email or other data which will be used by the application

The project team has also explored other packages for QR and barcode scanning, such as the Scan package. However, it is required that the barcodes should be in a specific orientation that can only scan one barcode out of each image. The quality of the mail piece image scan also impacts the ability of the scanner to detect the barcode.

### Text-to-Link Component

The Text to Link component “Linkifys” all the text. The package that will be used is called Link Well. All text from mail pieces or emails will be displayed via this component so embedded links, phone number and emails in the text will be selectable by the user. The project team also investigated other packages such as the linkify package, however since it accomplished the same thing as the Link Well package it was decided that the more popular package would be used for better maintainability.

### Voice Assistant Component

The Voice Assistant Component uses the native assistant applications on the user's phone to accomplish tasks without the need for the application to be in the foreground via voice or text input. The Voice Assistant Component will use Google Assistant for Android. It was decided against implementing Siri support for iOS due to development limitations.

#### Google Assistant

Google Assistant is an assistant application that comes installed on all Android devices. This application takes user queries and provides a solution for these queries. For example, a user could say “Hey Google, set an alarm for 10 AM” and Google Assistant would take that query and interact with the Clock application to create an alarm. This system can also be expanded by developers to provide support for non-system applications using App Actions. This is what will be used by the project team to implement Google Assistant support. This code will be native code as there is no flutter plugin for this functionality. The application will declare in its manifest what types of actions it can handle to inform Google Assistant of its capabilities. When Google Assistant receives a query for the application to do an action that it supports, it will then send an intent matching what Google Assistant thinks the user wants the application to do. For example, it would send the “Get Thing” intent if the user was searching for a mail piece. This intent would then include what the user said in string format. The application would parse the information from this intent to decide what it should do, then will run the specified action using the information the user performed. Since Google Assistant will only be sending the intents to the native application, we will use the receive\_intent plugin to send those intents up to the dart application where we can then use it to navigate to new pages.

While there were no other ways to implement this support, the project team did record some concerns about this approach. The application will be using Google’s built-in intents as they are more accurate and cover the user cases needed to implement this functionality. However, they limit what users can say to activate these actions compared to using custom intents. The project team looked at custom intents, and it was concluded that they would add unneeded complexity for minimal benefit. It also requires Google Play store support which will require the application being deployed on the Google Play store and passing review.

**5.3.5.1**  **Mail Piece Storage**

Mail Piece storage communicates directly with the Mail Fetcher service. Mail Fetcher will be calling the OCR whenever Digest emails are searched and running in the background to translate all content from text to QR codes. This data will be packaged into the JSON data models described in Section 4 of the TDD. This data will be cached in the device and the database. When the user selects a mail piece to view, it will retrieve the cached data and display it.

## Component Diagram

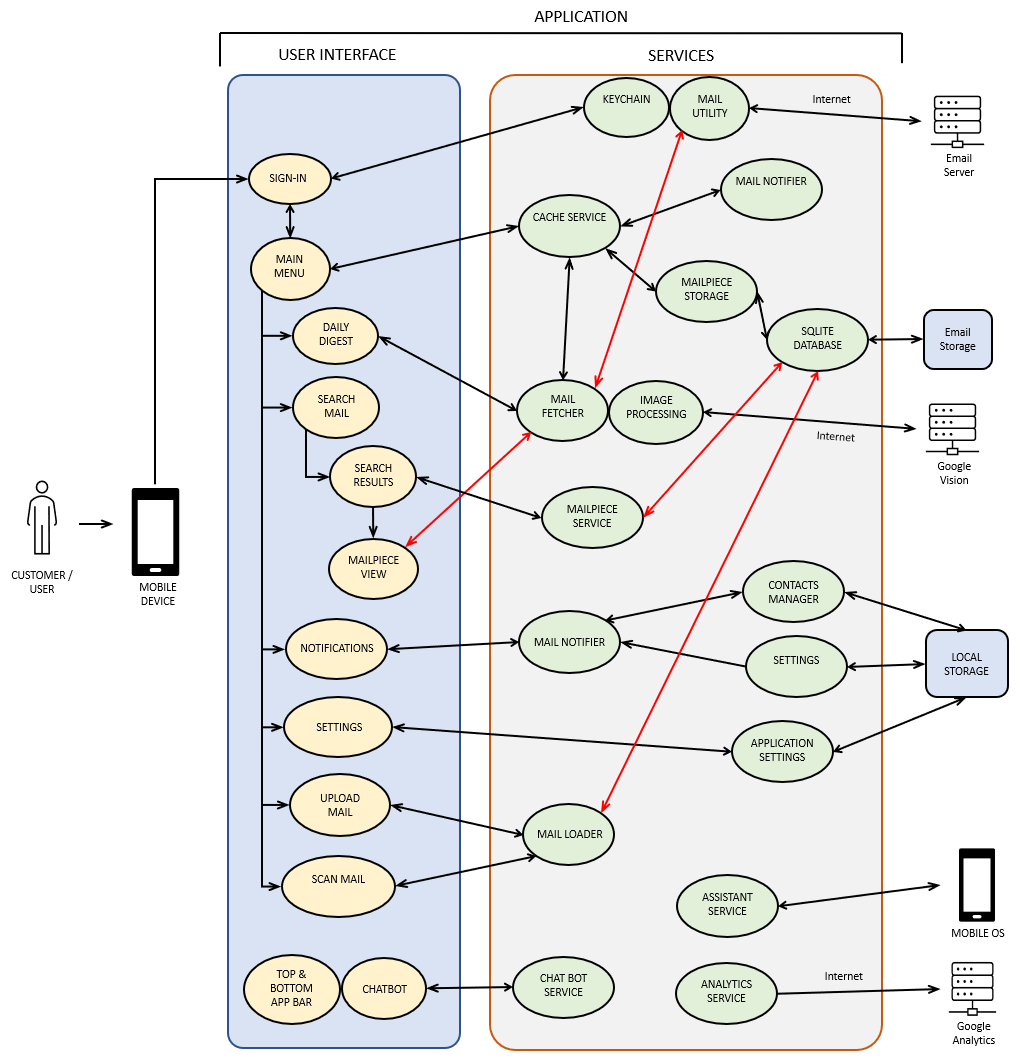


Figure 1A Component Designs – UI & Services

Text

Description automatically generated with medium confidence

Figure 1B Component Designs - Models

# Human Interface Design

The application User Interface (UI) will be developed using the Flutter development framework and the Dart programming language. Google's Cloud Services Platform (GCP) 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 be able to press the mic button to request details, scan mail to aid the user verbally and inform them of what mail they received. In addition, the application can 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 sender.

## User Flow

After downloading and installing the application, the user's journey within the application begins with the system checking for cached email credentials. If this is their first time accessing the app, or if this is their first open after their credentials have been cleared, they will need to re-enter that information and complete a successful login event in order for the application to start serving the various features and interfaces that require access to their email. Once the user is past the login, they will be directed to the Main Menu, which is a staging point for the entire application. At this point, all features are accessible to the user until they select the logout and their credentials are cleared.

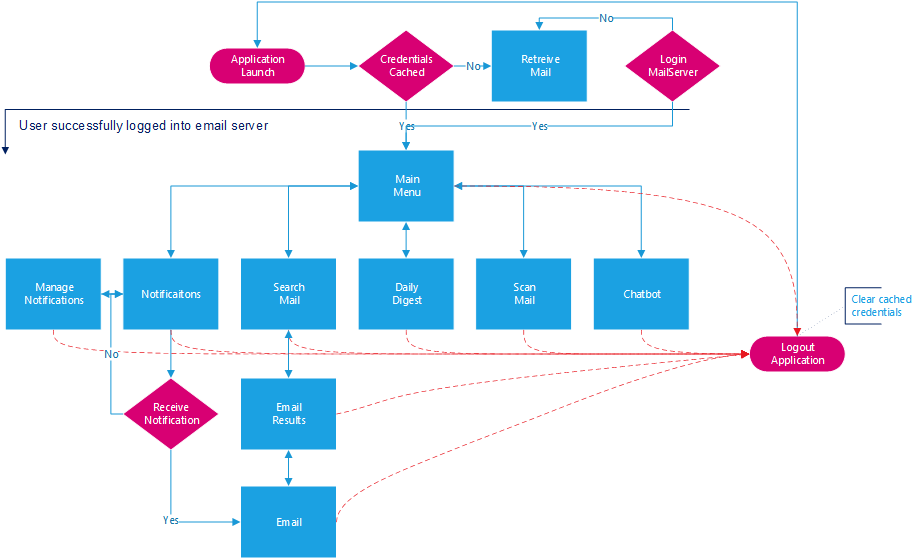


Figure 2 Application User Flow

## Screen Images

The following mock-up designs are each accompanied by an explanation of the user experience when interacting with that component.

* Mail login credentials screen
* Main user interface screen
* Application settings screen
* Search interface screen
* Search results screen
* Email display screen
* Multiple contacts identified in email
* In-App notifications screen
* Manage notifications screen
* Chatbot interface screen

### Mail login credentials screen

In the following user interface, the screen shall display an email address and password field used to retrieve email from the user’s registered USPS Informed Delivery registered email address. The user is also presented with a link to register for the USPS Informed Delivery service if they are not already signed up, and an additional link is displayed to access the USPS privacy policy.

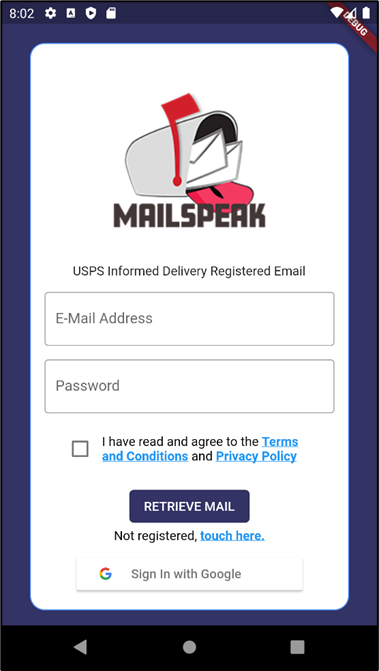


Figure 3 Retrieve Email UI

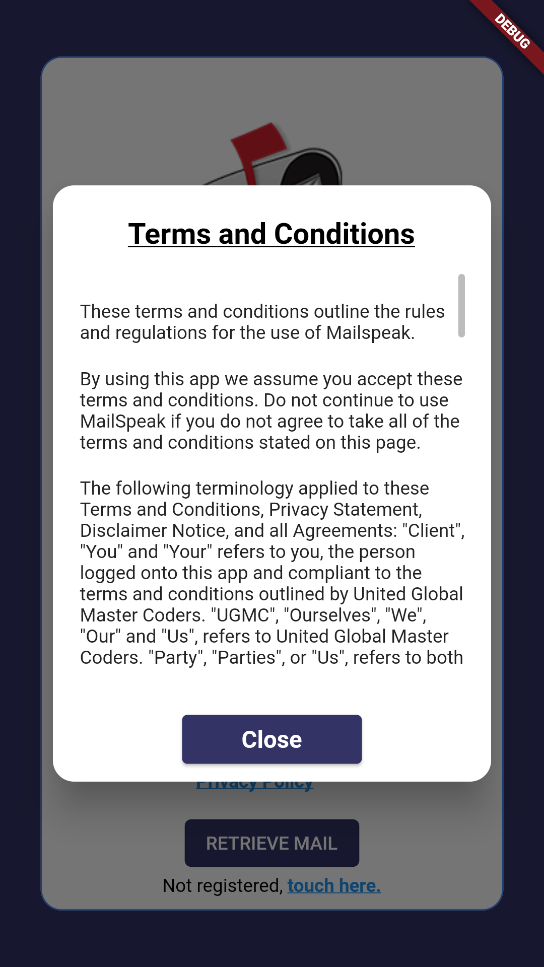


Figure Terms and Conditions Modal

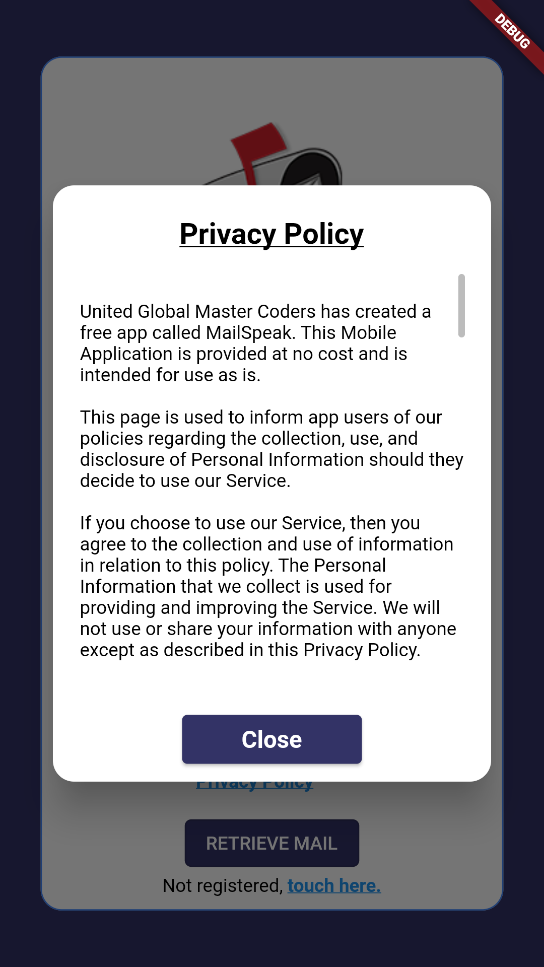


Figure Privacy Policy Modal

### Main user interface screen

The screen shall make the main application features available to the user in the following user interface. At the top right of the interface, the user can access the chatbot feature, which carries through the entire application. The center of the interface provides access to the search, daily digest, scan mail, upload mail, chatbot, and notification features which navigate the user to a different part of the application. The final navigational element is the bottom bar, which provides access to the main menu, plus the three main email features of the application and settings.

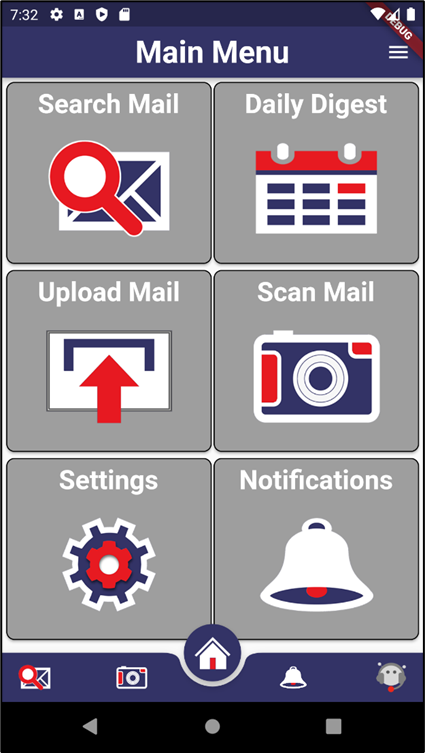


Figure Main Menu UI

### Application settings screen

In the following user interface, the user will be able to control the settings used by the application for converting image content into text. In addition, the user can control the level of detail used during detection and display.

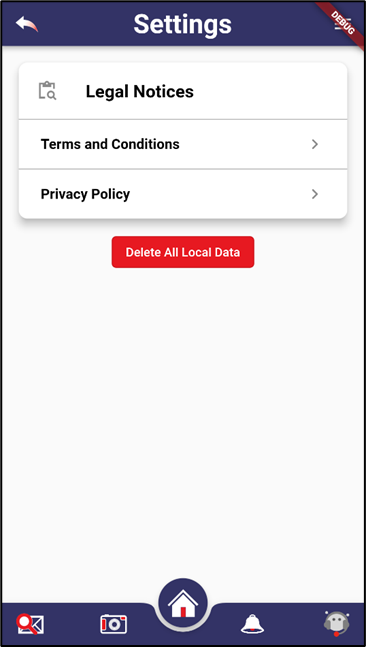


Figure Settings UI

### Search interface screen

The user must determine a period when searching for an email in the following user interface. 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 has the capability of autocomplete. As the user enters a value, the system will automatically provide logical assumed values and provide those for the user to select or continue entering letters until the autocomplete comes up with the correct word. This field is not required but can be valuable when there are many returned messages.

Graphical user interface, application

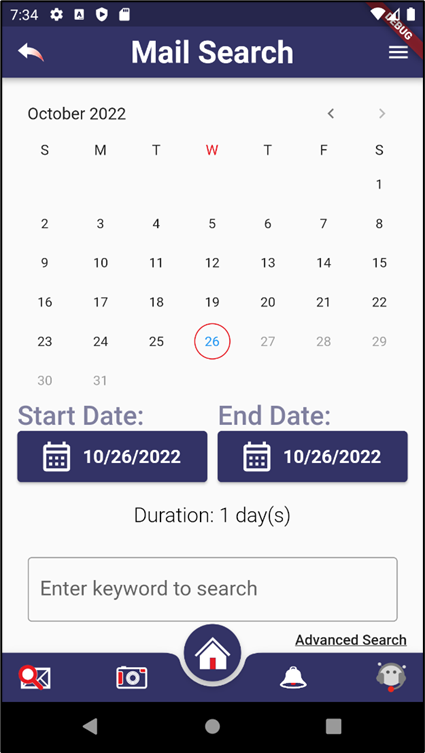
Description automatically generated

Figure Search UI

### Search results screen

In the following user interface, the search option remains at the top of the interface and is available for the user to refine the results. The returned results are displayed chronologically below the search pane in descending order.

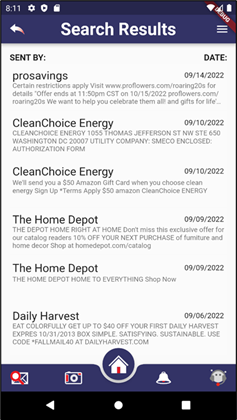
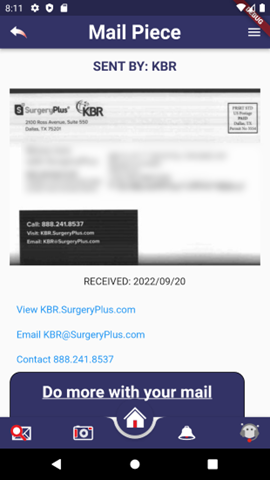


Figure Search Results UI

### Individual Mailpiece display screen

An email is displayed in a scrolling view in the following user interface. In the example provided below, a USPS Informed Delivery message will be displayed with a Subject, Sender, time received, and the message body. Each scanned physical mail will be followed by a section containing all of the sender information to provide access to the advanced call, email, text, and navigate to URL features of the application. Next, the Do more with your mail will be left intact and available for app users, and the image will follow the text content of the message. Additionally, there will be a clickable link in the content that the OCR parsed from any QR/Barcode/URL. This will be repeated for each physical mail image embedded in all USPS Informed Delivery email messages.

 Figure Email Display UI

### In-App notifications screen

While using the application, a user may experience an in-app notification. These alerts/notifications are triggered whenever an email is received that contains a positive match for a keyword alert that is set in the system. These alerts are either triggered against physical mail read by Google Vision or email fields. The notification will appear at the top of the window directly beneath the top header bar. The user may dismiss it by swiping it or tapping on the go-to message to be taken to the email that triggered the alert.

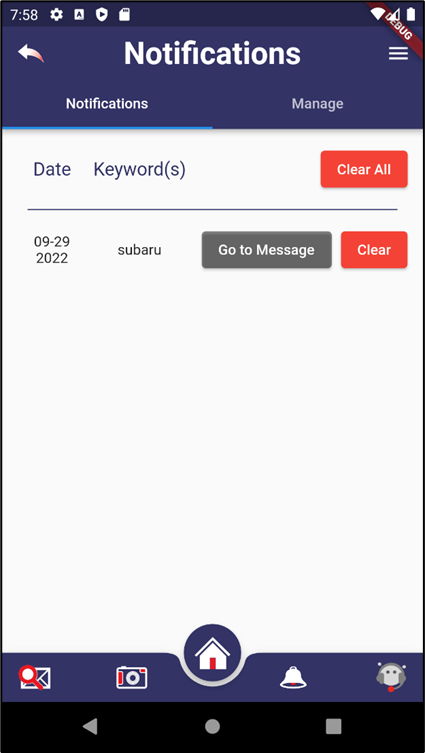


Figure Notifications UI

### Manage notifications screen

In the following user interface, the system displays the list of keyword alerts that have been added to the system. Entries are cleared by tapping the Clear button to the right of each line. New alerts are added by tapping the Add button, which brings up a new line for the user to enter in a value. Once the entry is correct, the user taps on save to complete the process.

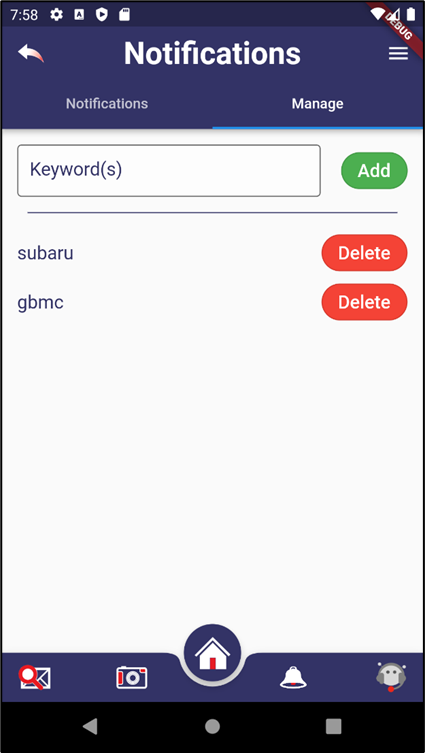


Figure Notifications Manager UI

### Chatbot interface screen

Application users can use the following interface to get answers to their questions or get assistance with tasks. Once the user taps on the chatbot icon, they are prompted with a question that begins the dialogue. Through a set of opening questions and the user responses, the system begins to narrow down the task and can respond to the user’s request.

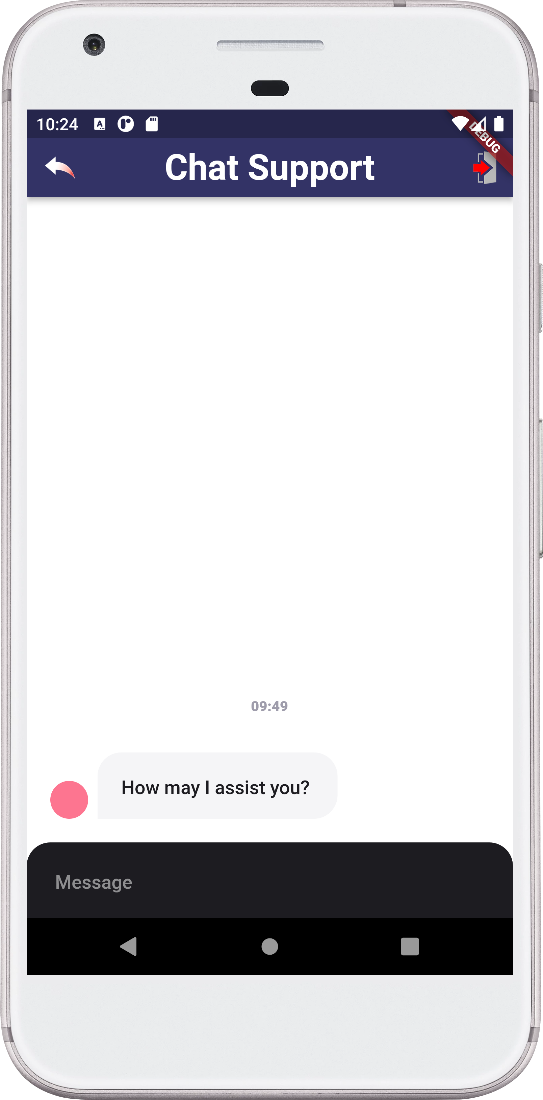


Figure Chatbot UI

# Requirement Matrix

The requirement matrix below lists the Mailspeak Application’s Mandatory and optional requirements. In addition, the requirement matrix also identifies which team oversees each requirement.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Requirement Description** | **Category** | **Team** |
|  | **Mail View** |  |  |
| 1 | Open the resulting email [from search} | Mandatory | A |
| 2 | Call, email, or send a text message to the sender or contact on command | Mandatory | A |
| 3 | Visit links or barcodes | Mandatory | A |
| 4 | Provide feedback to USPS on the links or barcodes visited | Mandatory | A |
|  | **Notification View** |  |  |
| 5 | Look out for email from someone | Mandatory | A |
|  | **Internal Feedback** |  |  |
| 6 | Research cyclic consumer behavior to report to USPS and its customers | Mandatory | A |
|  | **Voice Assistant** |  |  |
| 7 | Integrate with Google Assistant | Optional | A |

Table 4 Requirement Matrix

# Appendices

## Definitions, Acronyms, Abbreviations

This document contains several words that pertain to the application that is being developed. To give more clarity, however, the following terminology relating to the application is defined:

|  |  |
| --- | --- |
| Abbreviation | Definition |
| API | Application programming interface |
| JSON | JavaScript Object Notation |
| ML | Machine Learning |
| OCR | Optical Character Recognition |
| QR | Quick Response |
| SDK | Software Development Kit |
| UI | User Interface |
| USPS | United States Postal Service |

Table 5 Abbreviations, Acronyms, Definitions Table

## References

TeamArch-TTD for Reference: <https://umgcdev361.sharepoint.com/:w:/r/sites/SWEN670Fall2022/Shared%20Documents/General/Previous%20Class%20Docs%20(For%20Reference)/TDD/TeamArch-Technical-Design-Document.docx?d=wdac92747283044cc882b98e477d3abe3&csf=1&web=1&e=aThrxL>