Technical Design Document

usps informed delivery app – visually impaired

**Presented by: Team Arch: Reshawna Sampson, Barry Gartrell, Sheena Johnpeter, Stanley De Jesus, Ananya Srinivasan, Arnold Detoito**

2022

Table of Contents

[1 INTRODUCTION 3](#_Toc110975721)

[1.1 Purpose 3](#_Toc110975722)

[1.2 Overview 3](#_Toc110975723)

[1.3 Scope 3](#_Toc110975724)

[1.3.1 In Scope 3](#_Toc110975725)

[1.4 Definitions, Acronyms, and Abbreviation 4](#_Toc110975726)

[2 SYSTEM OVERVIEW 4](#_Toc110975727)

[3 ARCHITECTURAL DESIGN 4](#_Toc110975728)

[3.1 Decomposition Description 5](#_Toc110975729)

[3.1.1 Component 1: Image processing to detect logos, typed text, handwritten text 5](#_Toc110975730)

[3.1.2 Component 2: QR and Barcode processing 7](#_Toc110975731)

[3.1.3 Componenet 3: Address Validation 8](#_Toc110975732)

[4 DATA DESIGN 8](#_Toc110975733)

[4.1 JSON Request 9](#_Toc110975734)

[4.2 JSON Response 9](#_Toc110975735)

[4.2.1 Text Detection 9](#_Toc110975736)

[4.2.2 Logo Detection 14](#_Toc110975737)

[4.3 Application Data Model 15](#_Toc110975738)

[5 COMPONENT DESIGN 17](#_Toc110975739)

[5.1 Image–to–Text Component 17](#_Toc110975740)

[5.2 QR–Barcode Component 17](#_Toc110975741)

[5.3 Address Validation 17](#_Toc110975742)

[6 HUMAN INTERFACE DESIGN 18](#_Toc110975743)

[6.1 Overview of User Interface 18](#_Toc110975744)

[6.2 Screen Images 18](#_Toc110975745)

[6.2.1 Main Window 19](#_Toc110975746)

[7 REQUIREMENT MATRIX 24](#_Toc110975747)

[8 APPENDICES 26](#_Toc110975748)

[8.1 References 26](#_Toc110975749)

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision Number** | **Date** | **Description** | **Approved by** |
| 1.0 | 6/14/2022 | Initial document | Reshawna Sampson (Project Manager) |
| 1.1 | 6/17/2022 | Revised requirements matrix | Reshawna Sampson  (Project Manager) |
| 1.2 | 7/29/2022 | Updated images and tables to be APA. Cited in the references. | Reshawna Sampson  (Project Manager) |

# **INTRODUCTION**

## **Purpose**

The purpose of this Technical Design Document is to provide a comprehensive look at the design and architecture of the USPS Informed Delivery App that assists visually impaired users with accessing this service. The sections of this document will outline the application’s architectural design, the data design, and the component design. This document is intended to be used by Software Developers/Testers, Project Managers, Stakeholders, Business Analysts, or any other person who could improve the application.

## **Overview**

The purpose of the USPS Informed Delivery App is to assist visually impacted users in accessing the USPS Informed Delivery Service. USPS Informed Delivery is a service that will email a person what is coming to them in the mail. For the Application to deliver this information to the user, the application will need to use the user’s email address to gather the information for the user’s Informed Delivery digest. This Application will take the images from the email, retrieve the text using the Google Vision API, and then have the information read to the user. Additionally, the user will be able to scan QR codes. Team Arch is tasked with working on the image processing with the Google Vision API, the QR code reader functionality, and the processing of an individual image scan.

## **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 functionality of the app.
* **Data Design:** This section will detail the data that the system will utilize for the app to function.
* **Component Design:** This section will detail the components of the app.
* **Human Interface Design:** This section will detail the functionality of the human interface design of the USPS Informed Delivery App.

## **Definitions, Acronyms, and Abbreviation**

The below table defines most used acronyms/abbreviations in this TDD document.

**Table 1**

*Acronyms, Abbreviations, and Definitions*

|  |  |
| --- | --- |
| Word/Abbreviation | Meaning |
| TDD | Technical Design Document |
| TBD | To be determined |
| APP | Application |
| UI/UX | User Interface |
| API | Application Programming Interface |
| GCP | Google Cloud Platform |
| GUI | Graphical User Interface |
| iOS | iPhone Operating System |
| OCR | Optical Character Recognition |
| QR | Quick Response |
| USPS | United States Postal Services |
| JSON | JavaScript Object Notation |
| ML | Machine Learning |

# **SYSTEM OVERVIEW**

Team Arch’s main goal is to create the USPS Informed Delivery Mobile application, which will inform visually impaired individuals about what is in their mailboxes. This application will also be able to inform visually impaired individuals when a daily digest email arrives, provide an audio explanation of their Informed Delivery Daily Digest emails, and deliver images of mails using the Google’s Cloud Services Platform (GCP). USPS Informed Delivery Mobile application will mainly include cloud-based technologies for image-to-text, logo identification, and support for the QR code/barcode reader functionality (TBD). In addition, the application will analyze emails for logos and offer further information about the sender. Figure 1 below illustrates the flow of data (the encoded image) through the GCP and two major functions that are being used.

# **ARCHITECTURAL DESIGN**

The architectural design depicts the back-end components of the USPS Informed Delivery App. It also shows the flow of data from retrieving the data to the output of the returned data. The back-end components include image processing to detect handwritten text, logos, and typed text, QR/ barcode processing components (TBD), and address validation using USPS Web Tools API. Google Cloud Platform is being used to encompass most of the architecture behind the USPS Informed delivery app.

Figure 1 depicts the full system architecture including all the major components.

**Figure 1**

*Full System Architecture*

Chart, scatter chart

Description automatically generated

## **Decomposition Description**

When the user opens the USPS Informed Delivery App, 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 back-end will be highlighted.

### **Component 1: Image processing to detect logos, typed text, handwritten text**

The image processing component consists of API calls to the Google Cloud Vision API. The image is encoded using base64 format and added to the request JSON under content as a String. The request JSON is then sent directly to the Google Cloud for image processing. Cloud Vision will then send back a response JSON with the outcome. Figure 2 and 3 below illustrates the major components of the GCP for this architectural design. See section 4 for details.

**Figure 2**

Flow of data for GCP

Application

Description automatically generated

**Figure 3**

Sequence Diagram of GCP

Diagram

Description automatically generated with low confidence

### **Component 2: QR and Barcode processing**

For this requirement, Team Arch is currently researching and analyzing two viable options and approaches. There is an open-source Flutter package scan, and a Google Cloud Platform option called Google MLKit. The following subsection will give more detail on each option.

#### **Scan - pub.dev package**

Pub.dev is an official package repository for Flutter and Dart applications. This website repositor is also supported by Google. The scan package has the ability to use the ScanView in a Flutter widget tree to show the camera scan lines specifically for scanning QR codes and bar codes. It then uses the “scan.parse” built-in functionality to decode the QR code from an image.

As of the creation of this document, two reasons why this has not yet been chosen:  it is difficult for the canner to read low-quality/resolution codes, and it only works in a specific orientation: portrait mode.

#### **Google ML Kit**

Google ML Kit is a mobile SDK utilized for on-device machine learning tasks for Android and iOS apps. ML Kit is primarily utilized for image recognition tasks, such as face detection and barcode scanning. The corresponding Flutter plugin for the barcode scanning API is called google\_mlkit\_barcode\_scanning. An instance of the BarcodeScanner class is created, and an image object or byte array is passed into the scanner instance’s *processImage()* method. The method returns the number of barcodes (1D or 2D) detected and their corresponding links or values.

### **Componenet 3: Address Validation**

This component will connect to the USPS Web Tools API Library using http requests and xml. This will be used to validate the addressed being returned by the Google Vision API response. For credentials to use the USPS Web Tools API an account needs to be created in order to have access to the numerous APIs. Once the account is created, a username and password are generated to be used to access the API. The one this component will be using is the <AddressValidateResponse> . The follow xml examples show how the request is made using their production or secure shipping API call, and the response that is returned.

**Request:**

http://production.shippingapis.com/ShippingAPI.dll?API=Verify &XML=<AddressValidateRequest USERID="xxxxxxx"><Address ID="0"><Address1></Address1> <Address2>6406 Ivy Lane</Address2><City>Greenbelt</City><State>MD</State> <Zip5></Zip5><Zip4></Zip4></Address></AddressValidateRequest>

**Response:**

<?xml version="1.0"?> <AddressValidateResponse><Address ID="0"><Address2>6406 IVY LN</Address2><City>GREENBELT</City><State>MD</State><Zip5>20770</Zip5> <Zip4>1440</Zip4></Address></AddressValidateResponse>

# **DATA DESIGN**

Data shall be transmitted and received using JavaScript Object Notation (JSON). JSON objects invoke requests and receive responses to and from the Cloud Vision API. 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. See table 2 for a description of each and figure 4 for the request JSON.

**Table 2**

*Project Features Type*

|  |  |
| --- | --- |
| **Features Type** | **Description** |
| TEXT\_DETECTION | Performs OCR to convert any text within an image. |
| LOGO DETECTION | Detect company logos within the image. |

**Figure 4**

*Request JSON*

Graphical user interface, text, application

Description automatically generated

## **JSON Response**

### **Text Detection**

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. See figure 5 for the image provided for image processing and how it was split into those refined objects. Figure 6, 7 and 8 as an example JSON response for the images shown in figure 5.

**Figure 5**

Sign – Image Processing results for fullTextAnnotation Objects/breakdown

A blue sign with white text

Description automatically generated with medium confidenceA picture containing text, outdoor, sign

Description automatically generated

A picture containing text

Description automatically generated

**Figure 6**

Sign Image JSON response

**Figure 7**

Sign Image JSON response

Table

Description automatically generated with low confidence

**Figure 8**

Sign Image JSON response

Graphical user interface

Description automatically generated with low confidence

### **Logo Detection**

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. See Figure 9 for a sample Cloud Vision JSON response for logo searching.

**Figure 9**

Logo Response

A picture containing chart

Description automatically generated

## **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 3 for a description of each attribute.

**Table 3:**

JSON object attributes for UI team

|  |  |  |
| --- | --- | --- |
| **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 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 find more information |

# **COMPONENT DESIGN**

As described in the Architectural Design section, the application backend consists of the following components: Image-to-Text and QR-Barcode Scanning, QR-Barcode Component, Address Validation. Pertinent data extracted from the mail piece image, such as sender information and affiliated links, is returned in a JSON format.

## **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 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 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. The quality of the mail piece image scan also impacts the ability of the scanner to detect the barcode.

## **Address Validation**

USPS Web Tools API is a library set of APIs that are primarily used by companies shipping packages. For the address validation the <AddressValidationResquest> will be used. Once the API request is sent using http request the response is returned. This is formatted in xml to make the right request.

# **HUMAN INTERFACE DESIGN**

## **Overview of User Interface**

The application User Interface (UI) will be developed using the Flutter development framework and the Dart programming language. The 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 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 sender.

## **Screen Images**

The following screen images illustrate the mockup views of the main aspects of the application UI, along with textual descriptions of their purpose and contents. The list of screen images is as follows:

* Main Window
* Latest Digest Window
* Latest Email Window
* Scan Mail

Below discusses the detail use cases, action in the interfaces for each screen.

### **Main Window**

**Screen Name:** Main Window

**Figure** **10**

*Digest Mode*

Graphical user interface, text, application

Description automatically generated

**Internal UI functionality**:

“Latest”Button – When a user clicks the “Latest”button or says “latest digest”, it opens up the latest email and automatically sends it to Google's Cloud Services Platform (GCP) for processing to read the email content to the user and check for any artifacts (QR codes, logos, etc.).

“Scan Mail” Button – When a user clicks the “Camera” button, it opens the camera window and enables camera to scan the mail to audibly read sender/recipient and other information found in the generated JSON file.

**External UI functionality**:

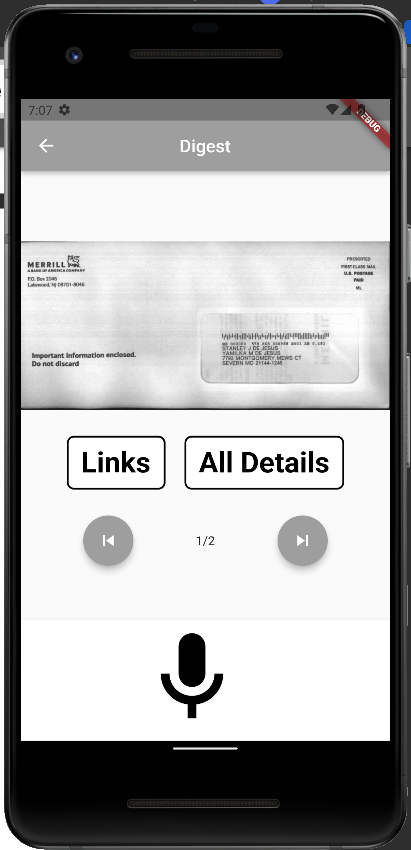
N/A

#### **“Latest Digest” Window**

**Screen Name:** Latest Digest Window

**Figure 11**

*Digest Page*



**Internal UI functionality**:

The application locates the latest email, it sends it to Google's Cloud Services Platform (GCP) OCR features for processing to check for text and any artifacts (QR code, logos, etc.), then it creates a JSON file and sends it to UI to craft messages and audibly read sender/recipient and other information found in the generated JSON file.

**External UI functionality**: N//A

##### **Latest Menu Window**

**Screen Name:** Latest Email Menu Window

**Figure 12**

*Email Mode*

Graphical user interface, text, application

Description automatically generated

**Internal UI functionality**:

When a user clicks on the application screen, the application displays the mail menu on top of the mail. The user will have the button options to interact with the application.

“Previous” Button – When a user clicks the “Previous” button, it updates the mail screen with the previous email that is stored.

“Next” Button – When a user clicks the “Next” button, it updates the mail screen with the stored next email.

**External UI functionality**: N/A

#### **Email Page (Voice Command) Window**

**Screen Name:** Email Page (Voice Command) Window

**Figure 13**

*Email Page*

Text

Description automatically generated with medium confidence

**Internal UI functionality**:

The application changes the mic animation to display that the mic is active and start recording. The application Speech to Text converts to text, checks for command phrases and performs specific action or call the component of the application, see external UI functionality below.

**External UI functionality**: N/A

#### **Camera Window**

**Screen Name:** Camera Window

**Image**:

Text

Description automatically generated

**Internal UI functionality**:

The application turns on the camera screen to let the user scans the mail per the general mail guidance (sender – top left and recipient – center right), and sends the image to Google's Cloud Services Platform (GCP) to perform image processing and sends back results.  The application creates a JSON file with results, crafts message, converts from text to speech and plays audio.

**External UI functionality**: N/A

# **REQUIREMENT MATRIX**

The requirement matrix below lists the USPS Informed Delivery Mobile Application’s Mandatory, optional requirements. In addition, the requirement matrix also identifies which team oversees each requirement.

**Table 4**

Requirements Matrix

| **Code** | **Requirement** | **Team Arch**  **is responsible for** | **Team B is responsible for** | **Mandatory or Optional** |
| --- | --- | --- | --- | --- |
| R.01 | The application shall use the user’s email address to collect information for the user’s Informed Delivery Digest |  | 🗹 | Mandatory |
| R.02 | The application shall show a screen that includes the user’s personal information |  | 🗹 | Optional |
| R.03 | The application shall have access to  the voice command feature |  | 🗹 | Mandatory |
| R.04 | The application shall identify the user’s voice |  | 🗹 | Mandatory |
| R.05 | The application shall have access to the mic button for requesting the user’s information |  | 🗹 | Mandatory |
| R.06 | The application shall listen to the user’s voice when the user selects the mic button |  | 🗹 | Mandatory |
| R.07 | The application shall notify the user when a daily digest email arrives |  | 🗹 | Mandatory |
| R.08 | The application shall listen for keywords/phrases to interact with the users |  | 🗹 | Mandatory |
| R.09 | The application shall turn keywords/phrases into text in order to identify what actions should be taken |  | 🗹 | Mandatory |
| R.10 | The application shall have access to the user’s phone camera | 🗹 |  | Mandatory |
| R.11 | The application shall use Google Vision API OCR and logo features for image processing | 🗹 |  | Mandatory |
| R.12 | The application shall extract an image from the user’s email |  | 🗹 | Mandatory |
| R.13 | The application shall recognize handwritten content | 🗹 |  | Mandatory |
| R.14 | Upon recognizing handwritten contents, the application shall extract an image from the user’s email and verbally read the handwritten content out to the user |  | 🗹 | Mandatory |
| R.15 | Upon extracting an image of an email using the user’s phone camera, the application shall read the text aloud to the user |  | 🗹 | Mandatory |
| R.16 | The application shall detect logos from daily digest emails | 🗹 |  | Mandatory |
| R.17 | Upon checking for logos, the application shall provide additional information about the sender to the user | 🗹 |  | Mandatory |
| R.18 | The application shall have access to the QR/barcode reader functionality | 🗹 |  | Mandatory |
| R.19 | The application shall read QR/barcodes | 🗹 |  | Mandatory |
| R.20 | The application shall have a “Play Latest” feature and allow users to locate the latest email |  | 🗹 | Mandatory |
| R.21 | Data shall be transmitted and received using JSON format | 🗹 |  | Mandatory |
| R.22 | The application shall collaborate with iOS and Android Operating Systems | 🗹 | 🗹 | Mandatory |

# **APPENDICES**

## **References**

Dart packages. (n.d.). Retrieved from https://pub.dev/

Detect Labels. (n.d.). Retrieved from <https://cloud.google.com/vision/docs/labels>

Detect logos. (n.d.). Retrieved from <https://cloud.google.com/vision/docs/detecting-logos>

Detect text in images. (n.d.). Retrieved from <https://cloud.google.com/vision/docs/ocr>

Google. (n.d.). *Barcode scanning | ML kit | google developers*. Google. Retrieved from https://developers.google.com/ml-kit/vision/barcode-scanning?hl=en

*Google\_ml\_kit: Flutter Package*. Dart packages. (2022, May 2). Retrieved from https://pub.dev/packages/google\_ml\_kit

*Scan: Flutter Package*. Dart packages. (2021, December 22). Retrieved from https://pub.dev/packages/scan

USPS. (n.d.). *Web tools API portal*. USPS. Retrieved from <https://www.usps.com/business/web-tools-apis/>

USPS. (n.d.). *Development Guide for Application Programming Interfaces to USPS Web Tools Service*. Development Guide for USPS Webtools API Services. Retrieved June 16, 2022, from <https://www.usps.com/business/web-tools-apis/general-api-developer-guide.htm>

Purdue Writing Lab. (n.d.). *APA tables and figures // Purdue Writing Lab*. Purdue Writing Lab. Retrieved July 29, 2022, from https://owl.purdue.edu/owl/research\_and\_citation/apa\_style/apa\_formatting\_and\_style\_guide/apa\_tables\_and\_figures.html