University of Maryland Global Campus

Presented By:

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

PROGRAMMER GUIDE

USPS Informed Delivery App – Visually Impaired

Table of Contents

[1. Introduction 3](#_Toc110030215)

[1.1 Purpose 3](#_Toc110030216)

[1.2 Intended Audience 3](#_Toc110030217)

[1.3 Technical Project Stakeholders 3](#_Toc110030218)

[1.4 Definitions. Acronyms, and Abbreviations 4](#_Toc110030219)

[2. Why USPS Informed Delivery App is Needed 5](#_Toc110030220)

[3. Setting up the Development Application Environment 5](#_Toc110030221)

[3.1 Flutter and Dart 5](#_Toc110030222)

[3.2 Android Studio 7](#_Toc110030223)

[3.3 Setting up Android Emulator 9](#_Toc110030224)

[3.4 Setting up Android Device 10](#_Toc110030225)

[3.6 Why GitHub Desktop 20](#_Toc110030226)

[4. Application Development Code 20](#_Toc110030227)

[4.1 Code Repository 20](#_Toc110030228)

[4.2 Cloning the Code Repository 20](#_Toc110030229)

[4.3 Code branches 21](#_Toc110030230)

[5. USPS Informed Delivery App Architecture 22](#_Toc110030231)

[5.1 Application Architecture 22](#_Toc110030232)

[5.1 Design Pattern 23](#_Toc110030233)

[5.1.1 Object-oriented Programming 23](#_Toc110030234)

[6. Code Structure 23](#_Toc110030235)

[7. Development of USPS Informed Delivery App 25](#_Toc110030236)

[7.1 Development Languages 26](#_Toc110030237)

[7.2 Flutter Plugins and Libraries 26](#_Toc110030238)

[7.2 Implementing Google Cloud Vision API 28](#_Toc110030239)

[7.3 Implementing USPS Address Validation 37](#_Toc110030240)

[7.4 Implementing Digest Email Parsing/IMAP 41](#_Toc110030241)

[8. Testing 49](#_Toc110030242)

[8.1 Create a unit test 49](#_Toc110030243)

[9. Appendices 54](#_Toc110030244)

[9.1 References 54](#_Toc110030245)

|  |  |  |  |
| --- | --- | --- | --- |
| Revision Number | Date | Description | Approved by |
| 1.0 | 7/28/22 | Initial Document | Reshawna Sampson (PM) |
| 1.0 | 7/29/2022 | Added reference to APA. | Reshawna Sampson (PM) |
| 1.0 | 8/9/22 | Revised Images | Reshawna Sampson |

# Introduction

1.1 Purpose

The Programmer’s Guide documentation aims to describe the technical reasoning behind the development of the USPS Informed Delivery Vision Assistance App. Developers can use this document as a reference and guidance on implementing the different features of this mobile application. In addition to this document, there are a suite of documents that include the Project Plan, Software Requirements Specification, Technical Design Document, and Deployment and Operations Guide (Runbook.) that can be used in conjunction with this document.

1.2 Intended Audience

This Programmer’s Guide documentation is intended for the technical project stakeholders and future programmers who will continue developing this mobile application. It will provide detailed information on the implementation of the app to help assist in any new feature implemented.

1.3 Technical Project Stakeholders

The following table shows the technical project stakeholders for the USPS Informed Delivery Visual Assistance App:

**Table 1**

*Project Stakeholders*

|  |  |
| --- | --- |
| **Name** | **Role** |
| Dr. Mir Assadullah | Professor |
| Roy Gordon | Project Mentor |
| Robert Wilson | Project Mentor |
| Bereket Kelemu | Overall Project Manager |
| Reshawna Sampson | Team Arch Project Manager |
| Stanley De Jesus | Lead Developer |
| Barry Gartrell | Developer |
| Ananya Srinivasan | Business Analyst/Developer |
| Arnold Detoito | Developer/Tester |
| Sheena Johnpeter | Business Analyst/Tester |

1.4 Definitions. Acronyms, and Abbreviations

**Table 2**

*All Acronyms and Abbreviations*

|  |  |
| --- | --- |
| **Acronyms and Abbreviations** | **Definitions** |
| PM | Project Manager |
| BA | Business Analyst |
| UI | User Interface |
| OS | Operating System |
| iOS | iPhone Operating System |
| API | Application Program Interface |
| VM | Virtual Machine |
| CE | Code Editor |
| GCP | Google Cloud Services Platform |
| HI | Hardware Interface |
| IDE | Integrated Development Environment |
| SI | Software Interface |
| SDK | Software Development Kit |
| UML | Unified Modeling Language |
| UMGC | University of Maryland Global Campus |

# Why USPS Informed Delivery App is Needed

One of the challenges that visually impaired individuals face is dealing with vision loss or poor vision. People with impairments face a range of difficulties in their daily lives. Over 12 million Americans over the age of 40 experience some form of visual impairment (CDC.com). Smartphones are important in everyday life but using them for persons who are blind or visually impaired may be difficult or impossible to utilize.

The development of the cross-platform USPS Informed Delivery Vision Assistance Application, which will educate people with visual impairments about what is in their mails, is the main objective of Team Arch. To satisfy the expectations, the mobile application will also alert visually impaired people when a daily digest email comes, offer an audio explanation of their Informed Delivery Daily Digest emails, and transmit photos of mails via a user-friendly mobile application. People with an impairment will benefit from the speech recognition technology used in this mobile application.

# Setting up the Development Application Environment

3.1 Flutter and Dart

Dart is a free and open-source programming language that is designed for multiplatform application development. The Flutter framework is utilized in conjunction with Dart applications to create fast, natively compiled applications. For developing multiplatform applications, Flutter prevents the hassle of writing separate codebases for each platform, such as for Android and iOS. This saves development effort by basing the application off of a single codebase.

While the Dart SDK can be installed separately, the Dart SDK with CLI is also part of the Flutter SDK. The Flutter SDK installation varies depending on the OS, and there are multiple methods of installing Flutter, such as using Git or Homebrew. However, this guide will be covering how to install Flutter by downloading the SDK installation bundle.

For more details, please refer to the official Flutter documentation: <https://docs.flutter.dev/get-started/install>

**Windows**

*Environment Specifications:* Windows 10 or later (64-bit), x86-64 based, 1.64 GB disk space, Git for Windows 2.x installed

1. Download installation bundle (as of July 2022): <https://storage.googleapis.com/flutter_infra_release/releases/stable/windows/flutter_windows_3.0.5-stable.zip>
2. Extract the zip file and move the contained *flutter* folder to the desired installation location
3. To use Flutter commands in the Windows console, search for env in the Start search bar, and select “Edit environment variables for your account.” In “User variables,” check if a “Path” entry exists. If the path exists, append the path to *flutter\bin* using a semicolon separator. If the path doesn’t exist, create a new user variable called Path and append the path of *flutter\bin*. Close and reopen the console to check if the changes are in effect.
4. Run the *flutter doctor* command in the CLI to determine if any other dependencies need to be installed

**macOS**

*Environment Specifications:* macOS, 2.8 GB disk space, Git installed

1. Download the bundle that is optimized for the system processor (Intel or Silicon, as of July 2022):

* Intel: <https://storage.googleapis.com/flutter_infra_release/releases/stable/macos/flutter_macos_3.0.5-stable.zip>
* Silicon: <https://storage.googleapis.com/flutter_infra_release/releases/stable/macos/flutter_macos_arm64_3.0.5-stable.zip>

1. Extract the file to the desired location
2. To add Flutter commands to the console, open the *rc* file for the system shell (Bash or Z). The file to edit for Bash is *.bash\_profile* or *.bash\_rc*, and Z shell is *.zshrc*. Then, export the path to the Flutter directory.

Export PATH=”$PATH:[PATH\_OF\_FLUTTER\_GIT\_DIRECTORY]/bin”

And replace [PATH\_OF\_FLUTTER\_GIT\_DIRECTORY] with the directory path. Run *source $HOME/.<rc file>* to refresh the console (replace *rc file* with the rc file). To determine if the installation works, run the *which flutter* command.

1. Run the *flutter doctor* command in the CLI to determine if any other dependencies need to be installed

**Linux**

*Environment Specifications:* Linux (64-bit), 600 MB disk space

1. Download the installation bundle (as of July 2022): <https://storage.googleapis.com/flutter_infra_release/releases/stable/linux/flutter_linux_3.0.5-stable.tar.xz>
2. Extract the file to the desired location
3. To add Flutter commands to the console, open the *.bash\_rc* file. Then, export the path to the Flutter directory.

Export PATH=”$PATH:[PATH\_OF\_FLUTTER\_GIT\_DIRECTORY]/bin”

And replace [PATH\_OF\_FLUTTER\_GIT\_DIRECTORY] with the directory path. Run *source $HOME/.<rc file>* to refresh the console (replace *rc file* with the rc file).To determine if the installation works, run the *which flutter* command.

1. Run the *flutter doctor* command in the CLI to determine if any other dependencies need to be installed

**ChromeOS**

*Environment Specifications:* ChromeOS (64-bit) with Linux Beta turned on, 600 MB disk space

1. Download the installation bundle (as of July 2022): <https://storage.googleapis.com/flutter_infra_release/releases/stable/linux/flutter_linux_3.0.5-stable.tar.xz>
2. Extract the file to the desired location
3. To add Flutter commands to the console, open the *.bash\_rc* file. Then, export the path to the Flutter directory.

Export PATH=”$PATH:[PATH\_OF\_FLUTTER\_GIT\_DIRECTORY]/bin”

And replace [PATH\_OF\_FLUTTER\_GIT\_DIRECTORY] with the directory path. Run *source $HOME/.<rc file>* to refresh the console (replace *rc file* with the rc file).To determine if the installation works, run the *which flutter* command.

1. Run the *flutter doctor* command in the CLI to determine if any other dependencies need to be installed

3.2 Android Studio

Android Studio is an IDE specifically designed for developing Android applications. The latest version of Android Studio, as of July 2022, is Chipmunk. As Flutter is a framework designed for multiplatform application development, the Android Studio IDE is utilized for developing native Flutter applications for Android.

**Windows**

*Environment Specifications:* Windows 10 or later (64-bit), x86-64 based, 8 GB disk space, 8 GB RAM

*Package:*

* EXE (preferred): <https://developer.android.com/studio#downloads:~:text=android%2Dstudio%2D2021.2.1.15%2Dwindows.exe>
* ZIP: <https://developer.android.com/studio#downloads:~:text=android%2Dstudio%2D2021.2.1.15%2Dwindows.zip>

*Installation:*

1. If the exe file has been installed, double-click to launch

For a zip file, extract the zip file and copy the android-studio folder to the Program Files folder. Open the android-studio/bin folder and launch studio.exe or studio64.exe, based on machine

1. Follow the setup wizard and install the SDK dependencies

**macOS**

*Environment Specifications:* macOS Mojave or higher, 8 GB disk space, 8 GB RAM

*Package:*

* Mac 64-bit: <https://developer.android.com/studio#downloads:~:text=android%2Dstudio%2D2021.2.1.15%2Dmac.dmg>
* Mac 64-bit ARM: <https://developer.android.com/studio#downloads:~:text=android%2Dstudio%2D2021.2.1.15%2Dmac_arm.dmg>

*Installation:*

1. Launch the DMG file
2. Drag and drop Android Studio into the Applications folder, open the Applications folder, and launch Android Studio
3. Follow the setup wizard and install the SDK dependencies

**Linux**

*Environment Specifications:* Linux (64-bit), x86-64 based, 8 GB disk space, 8 GB RAM

*Package:*

* Linux 64-bit: <https://developer.android.com/studio#downloads:~:text=android%2Dstudio%2D2021.2.1.15%2Dlinux.tar.gz>

*Installation:*

1. Extract tar.gz file and move contents to where application files are stored
2. Open a terminal, change to the android-studio/bin directory, and execute studio.sh. This will launch the setup wizard
3. Follow the setup wizard and install the SDK dependencies

**ChromeOS**

*Environment Specifications:* ChromeOS (64-bit), 4 GB disk space, 8 GB RAM, Linux for Chrome enabled

*Package:*

* Chrome: <https://developer.android.com/studio#downloads:~:text=android%2Dstudio%2D2021.2.1.15%2Dcros.deb>

*Installation:*

1. Open the Files app and locate the DEB package that was downloaded in the Downloads folder under My files
2. Right-click the DEB package and select “Install with Linux (Beta)”
3. Follow the setup wizard and install the SDK dependencies

3.3 Setting up Android Emulator

**The significance of having an Android Emulator:**

For this project, our team decided to use the Android Emulator because it allowed our team members to easily test the application’s functionality efficiently across a range of Android API levels without having to own each individual physical device. Different Android Versions may be tested quickly and easily. Testing the application on an Emulator can be quicker and less complicated than testing it on a real device. The Android Emulator made it possible to evaluate how this application would perform on various devices, advise us of any modifications that would need to be made to the program’s functionality, enhance the user experience, and identify further areas for development.

**The instructions that follow explain how to set up an Android emulator:**

* For Windows installation of an Android emulator, follow the [**Android Emulator – Windows Installation Instructions.**](https://docs.flutter.dev/get-started/install/windows)
* For MacOS installation of an Android emulator, follow the [**Android Emulator – MacOS Installation Instructions.**](https://docs.flutter.dev/get-started/install/macos)
* For Linux installation of an Android emulator, follow the [**Android Emulator – Linux Installation Instructions.**](https://docs.flutter.dev/get-started/install/linux)
* For Chrome OS installation of an Android emulator, follow the [**Android Emulator – Chrome OS Installation Instructions.**](https://docs.flutter.dev/get-started/install/chromeos)

3.4 Setting up Android Device

To ensure that the application can run on an actual device we need to deploy it to an actual device. Deploying to an actual device will allow the developers to see how the application will function on different screen sizes and with different operating system versions. Also, this was a way for the team to be able to test the camera image scanning and QR/Barcode reading functionality. To set up an Android Device using Android Studio:

1. On the device, open the Settings app, select Developer Options, and then enable USB debugging (if applicable).
   1. This varies in different ways depending on the device, look into enabling developer options for your device.
2. Set up your system to [detect your device](https://developer.android.com/studio/run/device#:~:text=Set%20up%20your%20system%20to%20detect%20your%20device.).
   1. An example is that Windows needs to have a USB driver for ADB, install guide can be found [here](https://developer.android.com/studio/run/oem-usb).
3. Connect your device using a USB cable to your system.
4. Select Run on Android Studio and the application will install and open on your device.

3.5 Setting up an iOS Simulator & iOS Device (Mac Users)

An iOS Simulator needs to be set up to test and debug the application on an iOS platform. It mimics the application that runs on an iOS device. For the development of the USPS Informed Delivery App, we ha to simulate to an iOS device to test the “Scan mail “feature that utilizes the device camera. iOS simulators do not allow access to cameras.

**Setting Up an iOS Simulator**

1. Open iOS Simulator via Spotlight Search or run the following in a Terminal window or command line interface.

$ open -a Simulator

1. To select a different iOS device as a simulator, right-click on the Simulator app and select iOS device from the "Device" drop-down menu.

**Figure 1**

*Simulator Device*



One of the main features of the USPS Informed Delivery Visual Assistance App is to be able to scan an image of a letter and QR/Barcode using the user's phone camera. Since the iOS Simulator does not support camera functionality, the application needs to be deployed to a physical iOS device, such as an iPhone or iPad.

The image\_picker Flutter plugin is used for taking new pictures with the camera and selecting images from an iOS photo library. The image\_picker is added as a dependency in the pubspec.yaml file.

dependencies:

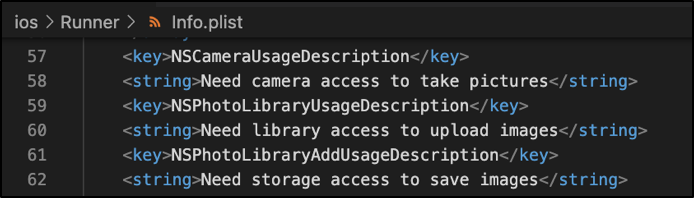
image\_picker: ^0.8.5+3

The user will need to explicitly grant access to the USPS Informed Delivery Visual Assistance App for it to use the iOS Camera app. In order for the user to grant access, the following <key> and <string> variables need to be added in the “Info.plist” file under the application's “ios/Runner” folder. These will make the "permission pop-up screen" appear the first time the user opens the application.

* NSCameraUsageDescription: A message that tells the user why the app is requesting access to the device’s camera.
* NSPhotoLibraryUsageDescription: A message that tells the user why the app is requesting access to the user’s photo library.
* NSPhotoLibraryAddUsageDescription: A message that tells the user why the app is requesting add-only access to the user’s photo library.

**Figure 2**

*Info.plist*



The user can also modify the permission settings for an application in the iOS device's Settings app, then the "Privacy" section.

The following are the steps to deploy the USPS Informed Delivery Visual Assistance App to an iOS device.

**Deploying the Application to a Physical iOS Device**

To deploy an application to a physical iOS device, a physical device deployment in Xcode needs to be set up. It is assumed that the user has an existing Apple ID which will be used as an Apple Developer account to access a limited set of developer-oriented privileges for free. Follow the Xcode signing process to provide an application.

1. Open the application's default Xcode workspace by doing one of the following steps.

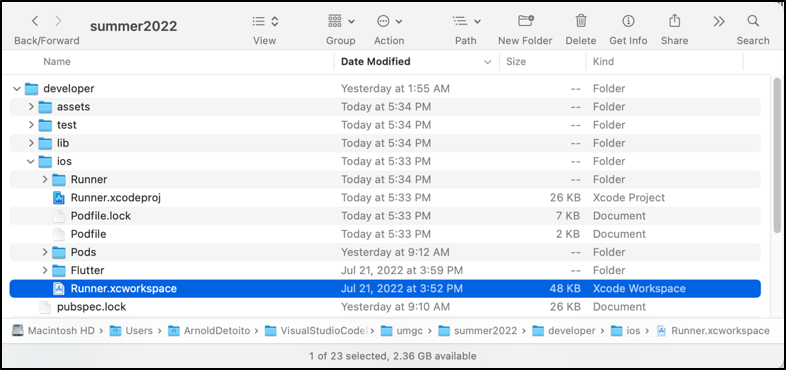
* Option 1: Run the "open ios/Runner.xcworkspace" in a Terminal window or command line interface from the application project directory to open the project.

$ open ios/Runner.xcworkspace

* Option 2: Double click the "Runner.xcworkspace" file from the application project "ios" directory to open the project.

**Figure 3**

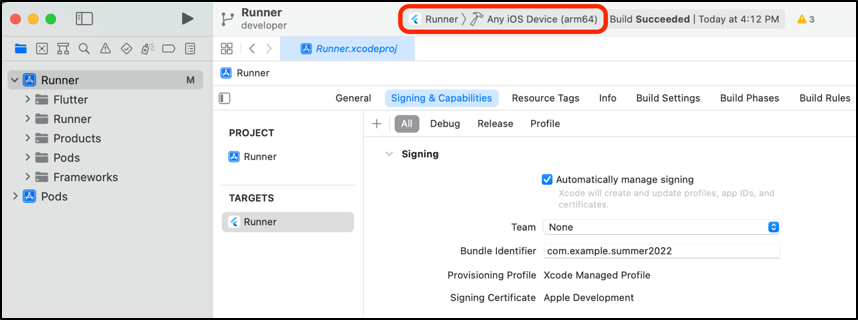
*iOS Directory*



1. Once the application project is opened in XCode, connect an iOS device using a USB cable and select an iOS device.

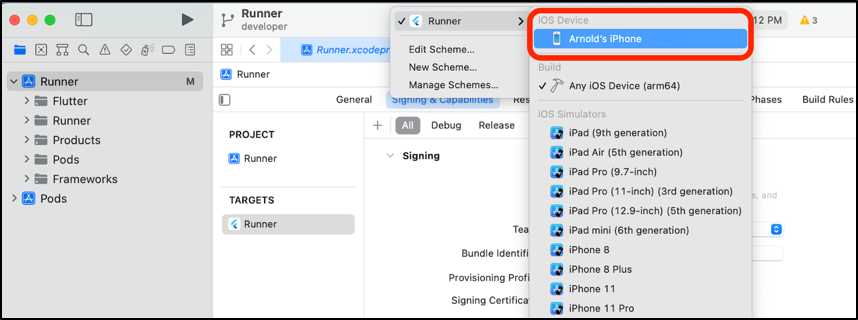
**Figure 4**

*XCode Application*



**Figure 5**

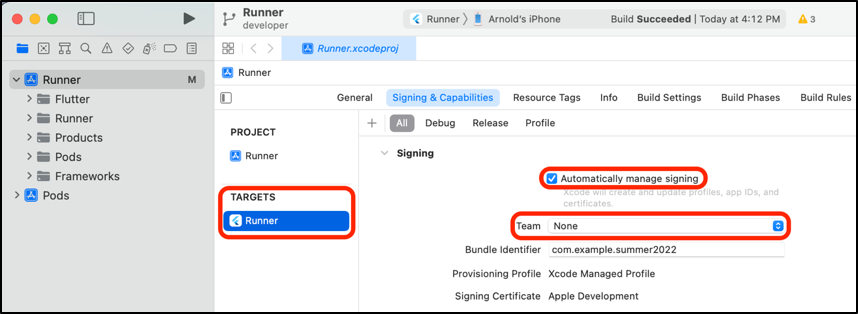
*XCode Application*



1. In the "Runner" Target settings page, select the Development Team in the "Team" section of the "Signing & Capabilities" tab and check the "Automatic manage signing" checkbox. This tells Xcode to create and download a Development Certificate, register a device with the account, and create and download a provisioning profile (if needed).

**Figure 6**

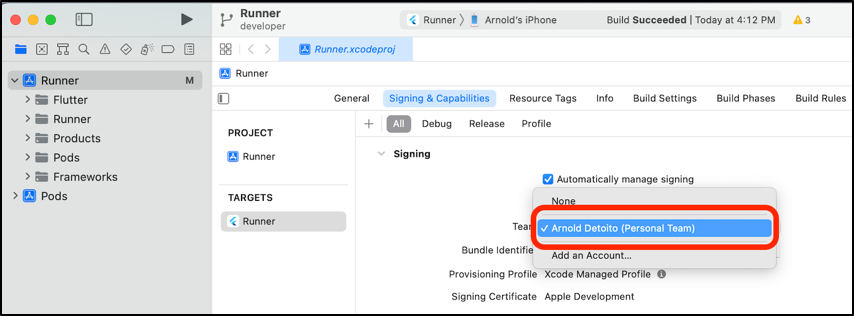
*Runner settings page*



If a user is already set up, select the account as the Development Team. Otherwise, select "Add an Account" to set up the user's Development Team account. An Apple ID will suffice for debugging and testing purposes.

**Figure 7**

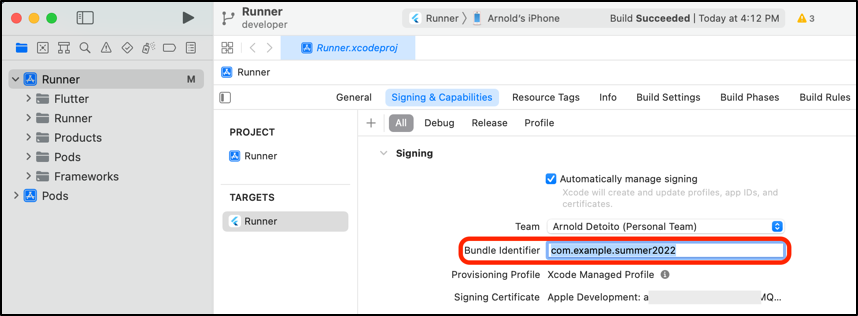
*Runner settings*



If there is a Bundle Identifier registration error, provide a unique bundle identifier, e.g. com.example.summer2022.

**Figure 8**

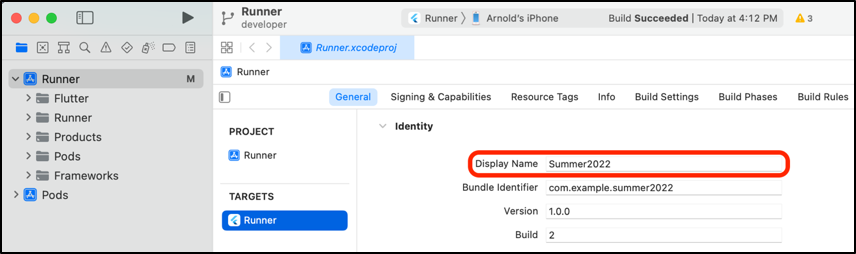
*Adding bundle identifier*



1. In the "Runner" Target settings page, if necessary, update the application's display name in the "Identity" section of the "General" tab.

**Figure 9**

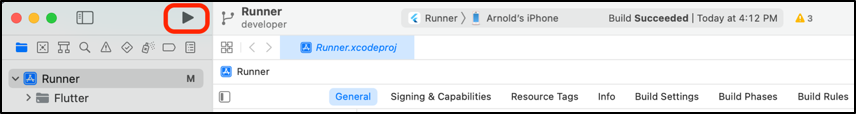
*Runner Settings General tab*



1. Click the "Run" button in XCode to build and deploy the application to the connected iOS device.

**Figure 10**

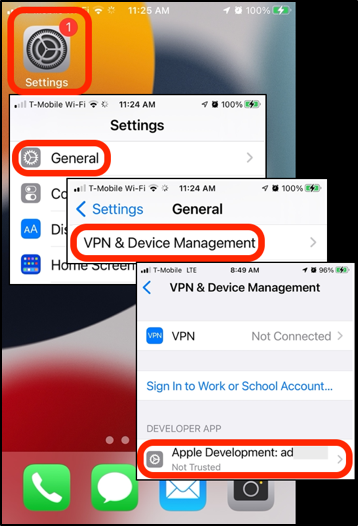
*Click Run*



1. If this is the first time the application is being deployed to this particular iOS device, the application needs to be trusted on that device. On the iOS device, go to the Settings app and navigate to the "General" section. From the "General" section, navigate to the "(VPN &) Device Management" section, then click on "Apple Development." under the “Developer App” and click “Trust Apple Development.” to trust the application.

**Figure 11**

*Settings General VPN & Device Management*



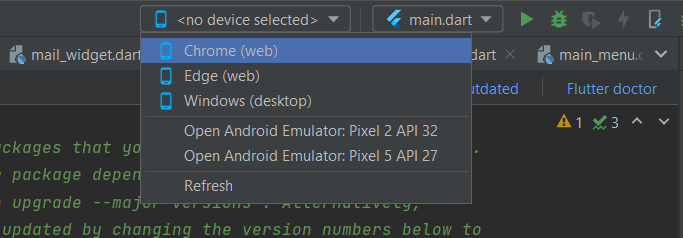
* 1. Test the development environment

To test the development environment: on Android Studios IDE.

Once Android studios have been installed and both flutter and dark SDKs are downloaded, click on the “no device selected” dropdown and choose “open android emulator”. An android phone emulator should load. Figure 12 depicts the no device dropdown and figure 13 shows the emulator once started.

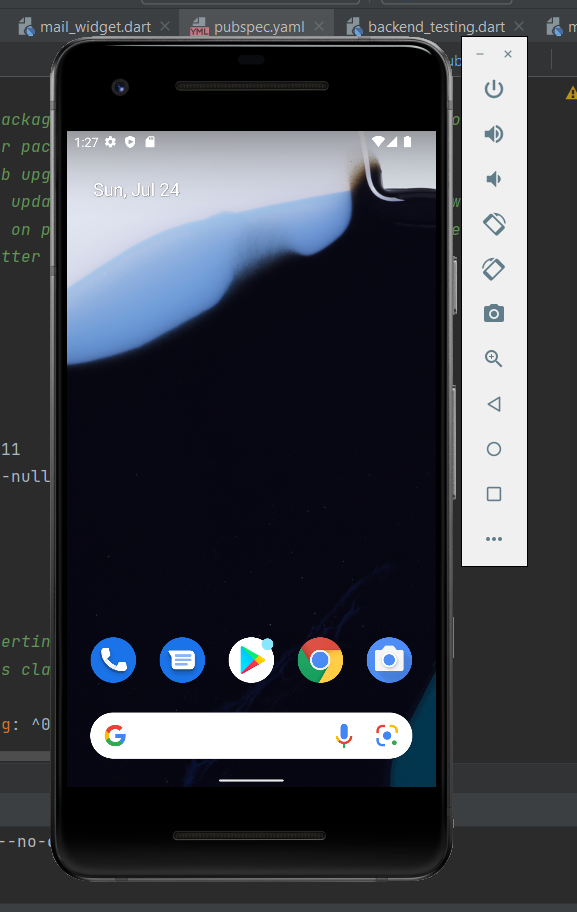
**Figure 12**

*Android Studio IDE*



**Figure 13**

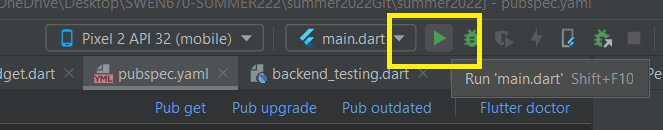
*Android Emulator*



On the IDE click “Run” the green arrow.

**Figure 14**

*Running the application*



The Futter application will begin building and the app should display on the android phone emulator. As depicted in the following image.

**Figure 14**

*USPS Informed Delivery App Main Page*

Graphical user interface, text, application

Description automatically generated

3.6 Why GitHub Desktop

GitHub Desktop is a very useful application that will allow a team to easily handle the source control for the application. This provides a developer with the ability to handle git commands in an easy-to-use user interface compared to using the command line to operate the more complex Git.

To install the application [Follow this documentation](https://www.softwaretestinghelp.com/github-desktop-tutorial/).

# Application Development Code

4.1 Code Repository

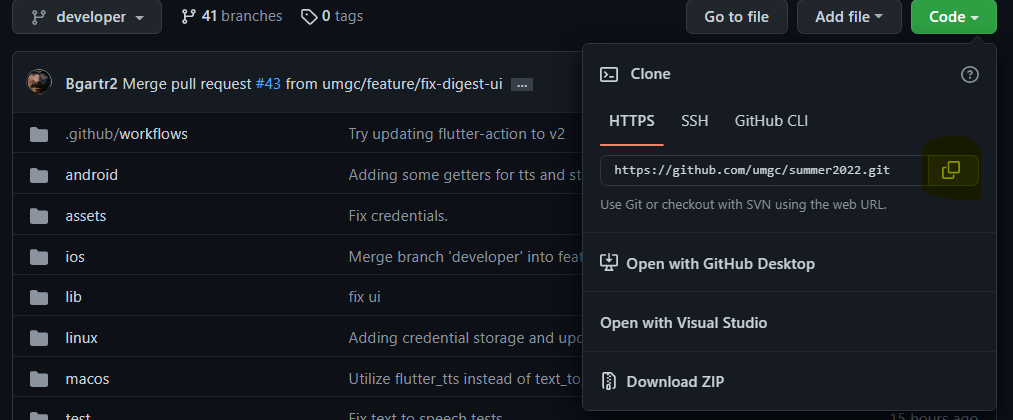
Our class repository is located on GitHub at <https://github.com/umgc/summer2022>. It can be accessed via the command line or through GitHub Desktop, which is a GUI approach to accessing the repository. GitHub is used for team collaboration providing version control, a place to download the most recent code version, and teams to manage and make changes at a central location. To use GitHub, you must have git installed. For more information on git, visit its main website at <https://git-scm.com/>.

4.2 Cloning the Code Repository

To clone GitHub, use the following Git command – git clone <git project URL>. The link can be obtained from the repository by clicking on the green button on the top right. A new window will pop up. Click on the copy icon highlighted in yellow to copy the URL.

**Figure 15**

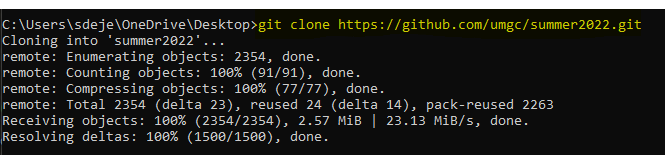
*Link to clone repository from GitHub*



On your command prompt, use the command below.

**Figure 16**

*Command to clone repository from command line*

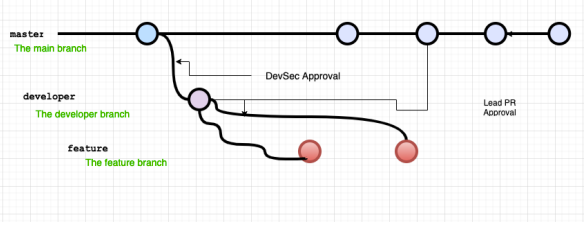


4.3 Code branches

The following image illustrates the flow and hierarchy of GitHub branches.

**Figure 17**

*GitHub Branches breakdown*



Our GitHub branch consists of types of branches.

* + Master – It is used to publish releasable code that will be released to both iOS and Android App stores. It also requires DevSec manager approval before updating this branch.
  + Developer – It is used for integrating features. It requires a minimum of two developers to approve push requests into this branch.
  + Feature – It is used by developers to test and code their features before integration.

The approach used is the bottom top (right to left if viewing the diagram) to ensure checks are performed before pushing updates to a higher branch. Before a feature branch is pushed to the developer branches, it goes through the Continuous Integration (CI)/Continuous Development (CD) process, which compiles the program and performs certain tests before it is allowed to be merged. CI/CD pipeline stores YAML files containing each environment's prerequisite steps. For this project, there are two environments that this project will support: Android and iOS. To push the developer branch into the main branch, GitHub will perform all checks that the developer has and requires the DevSec manager’s approval. This branch is used for deploying the application to the App Stores.

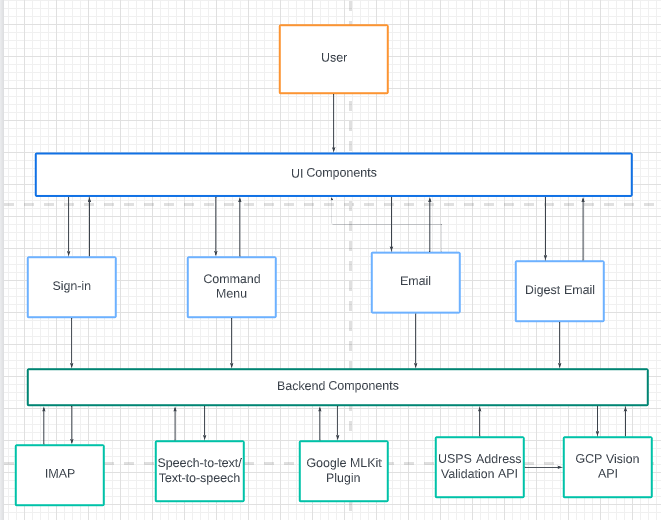
# USPS Informed Delivery App Architecture

5.1 Application Architecture

The application architecture is organized by a set of individual features interacting simultaneously. It is structured into two components: the UI and the backend. These two components can be further broken down into features. The application was structured this way because the class was split into two groups working on one application. To keep both teams working and not get bottlenecked, the features were identified and given to both teams. The UI section includes the UI design, the speech-to-text, text-to-speech, and connection to the user email using IMAP or POP3. The backed section includes the Google Vision API, Google ML Kit for barcode/QRcode reading, and USPS Web API for address validation. The Code structure section of this document has a visual of the folder layout in code. The following image depicts the structure.

**Figure 18**

*Architecture Structure*



5.1 Design Pattern

5.1.1 Object-oriented Programming

Object-oriented programming (OOP) was used to develop the USPS Informed Delivery Visual Assistance App. OOP is used to organize the software design based on the data or objects. This was chosen because of the OOP benefits of reusability and scalability. In the USPS Informed Delivery app this is showcased by the use of the MailResponse object that is created based on models of data that gets returned from Google Vision, the address validator, and the Google ML Kit.

# Code Structure

The USPS Informed Delivery Visual Assistance App was developed using the Flutter framework and the Dart language. The code structure for the USPS Informed Delivery Visual Assistance app can be broken down into the front-end (UI) and backend components. Like mentioned in section 5.1 of this documents these two components are further bokeh down based on features. The following image illustrates the file structure in code.

**Figure 18.1**

*Folder Structure*

Graphical user interface, text, application

Description automatically generated

**Figure 18.2**

*UI and Backend Folders by Features*

Graphical user interface, text

Description automatically generated

# Development of USPS Informed Delivery App

The tools used to create the USPS Informed Delivery Vision Assistance Application are described in the section that follows. It is divided into subsections that discuss the key elements of development.

**Development SDK**

**Flutter** is a Google open-source platform that allows individuals to create mobile, desktop, and online applications from a single codebase. Team Arch made the choice to utilize Flutter because it made it possible for us to create high-quality versions of this application rapidly and easily for iOS and Android without having to create separate code bases for each platform. Unlike other popular solutions, Flutter is a full SDK – *Software Development Ki*t – rather than a framework or library. In addition, Flutter is Google’s mobile application SDK for quickly creating high-quality native interfaces for iOS and Android, according to the official website (flutter.dev). This single code base is used to develop applications for several platforms, including Android and iOS. Both teams, Team Arch and B will be able to implement the USPS Informed Delivery Vision Assistance Application once and then deploy it for both Android and iOS devices.

7.1 Development Languages

As the language and runtimes that power Flutter, **Dart** will be utilized as the primary development language. Dart makes it possible to create cross-platform mobile applications efficiently. Due to its portability and flexibility, Team Arch chose to utilize the Dart programming language for this project. This simply implies that the USPS Informed Delivery Vision Assistance Application created by our team using Dart may operate without any restrictions on Android, iOS, Windows, MacOS, and other operating systems out there. It also primarily enables interactive elements of the program, including user widgets.

7.2 Flutter Plugins and Libraries

The Flutter plugins and libraries are published and available at pub.dev.

The following Flutter plugins and libraries are used to support the implementation of the USPS Informed Delivery Visual Assistance application.

***camera***

A Flutter plugin for iOS, Android, and Web allows access to the device cameras.

***cupertino\_icons***

A Flutter library that provides default icons for Cupertino widgets based on Apple-styled icons.

***enough\_mail***

A Flutter library that provides support for IMAP, POP3 and SMTP for email developers.

***flutter\_datetime\_picker***

A Flutter plugin that provides support for a date time picker for flutter.

***flutter\_secure\_storage***

A Flutter plugin to store data in secure storage.

***flutter\_svg***

A Flutter library that provides support for rendering Scalable Vector Graphics (SVG) files.

***flutter\_tt***

A Flutter plugin for Text to Speech. This plugin is supported on iOS, Android, Web, & macOS.

***global\_configuration***

A Flutter package for managing different configurations by merging them together and making them available everywhere inside the app via a singleton.

***google\_mlkit\_barcode\_scanning***

A Flutter plugin to use Google's ML Kit Barcode Scanning to read data encoded using most standard barcode formats.

***googleapis***

A Flutter library that provides support for accessing Google APIs described through the API discovery service.

***googleapis\_auth***

A Flutter package that provides support for obtaining OAuth2 credentials to access Google APIs.

***http***

A Flutter package that contains a set of high-level functions and classes that make it easy to consume HTTP resources. It's multi-platform, and supports mobile, desktop, and browser.

***image\_gallery\_saver***

A Flutter plugin project for saving images to the gallery, iOS needs to add the following keys to your Info.plist file.

***image\_picker***

A Flutter plugin for iOS and Android for picking images from the image library, and taking new pictures with the camera.

***intl***

A Flutter plugin for providing internationalization and localization facilities, including message translation, plurals and genders, date/number formatting and parsing, and bidirectional text.

***json\_annotation***

A Flutter plugin for defining the annotations used by json\_serializable to create code for JSON serialization and deserialization.

***json\_serializable***

A Flutter library that provides support for converting to and from JSON by annotating Dart classes.

***loader\_overlay***

A Flutter package to simplify screen management.

***path\_provider***

A Flutter plugin for finding commonly used locations on the filesystem. Supports Android, iOS, Linux, macOS, and Windows. Not all methods are supported on all platforms.

***path\_provider\_android***

A Flutter plugin for Android implementation of the path\_provider plugin.

***permission\_handler***

A Flutter plugin that provides support for a cross-platform (iOS, Android) API to request permissions and check their status.

***speech\_to\_text***

A Flutter plugin that exposes device-specific speech-to-text recognition capability.

***toggle\_switch***

A Flutter plugin for supporting the toggle switch widget. It can be fully customized with desired icons, width, colors, text, corner radius, etc. It also maintains a selection state.

***url\_launcher***

A Flutter plugin for launching a URL.

***xml***

A Flutter library for parsing, traversing, querying, transforming, and building XML documents.

7.2 Implementing Google Cloud Vision API

**Why Google Cloud Vision?**

There were different options for performing OCR/Logo image processing.  These options were Firebase ML Kit, Google ML Kit, and Google Cloud Vision. Firebase ML and Google ML kits were explored because there were walkthrough tutorials for implementing the feature. Unfortunately, these articles were dated 2021 or prior and did not function. At the time of research, Firebase ML Kit was depreciated and merged into Google ML Kit.  Google ML Kit did not provide accurate results, prompting me to look into our final option: Google Cloud Vision API. My first experiment was with Google Cloud Console, which allowed testing functionality and obtaining results without coding. The results were precise, but the generated outcome was extensive.

During the Google Cloud Vision implementation, it was not easy to obtain a connection using the google\_api package as it had minimal online tutorials and articles that could guide developers in using its functionality. Tutorials establishing a connection using Google's service account credentials were available but had some implicit quirks with obtaining a google cloud connection. While researching the google cloud connection, a GitHub thread was found where an example showed how someone got it to function.  The sample code was modified to meet our requirement of connecting to google service. Using multiple google service flutter tutorials, reading through forums, and finally, the Google Vision official tutorial site provided gleaned insight that was meshed to obtain a connection.  The other issue was the Google Vision API. There were many derivatives packages, and none of them worked. The decision came to use the google\_api official flutter package, which was not explicit about its use. A connection was established by tinkering with the API internal functionalities and using the JSON body shown below. The outcome was the same as the Google Cloud Console SDK. Flutter Google\_API does not support URLs but base 64 encoding images, which both Android and iOS also support. This was the approach to decide to stick to add the results were what we were looking for and manipulate to get our desired outcome.

**Google Vision API**

The following packages must be included in pubspec.yaml file under dependencies.

**Figure 19**

*Google Vision Package*



The "flutter pub get" command is executed next so that flutter can retrieve the required Google Vision API packages and store them for application use. There are two classes used in the Cloud Vision implementation: CredentialsProvoder and CloudVisionAPI.

**CredentialsProvider Class**

Once installed, the CredentialsProvider class is used to create a connection with Google Vision by using credentials stored in the credential file. The credential file can be obtained by following the Deployment Guide - Google Cloud Vision API setup (section 4.1) and storing it in the “assets” folder.  Then, the location is added in the pubspec.yaml file under assets to grant access to the file.

**Figure 20**

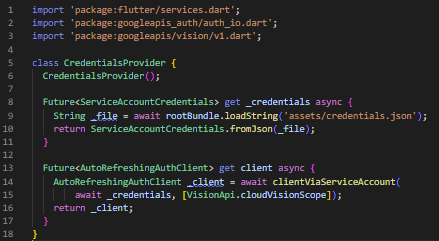
*Adding credentials to the project*



See below for the class implementation.

**Figure 21**

*CredentialProvider code implementation*



**CloudVisionAPI Class**

The developer does not interact with CredentialProvider class. Instead, it is used internally by the CloudVisionAPI class. The CloudVisionAPI consists of 3 public classes: search, searchImageForText, and searchForImageLogo.  See the below table for the method name and a brief description for each.

**Table 3**

*Public methods for CloudVisionAPI class*

|  |  |
| --- | --- |
| **Method** | **Description** |
| search | Calls searchImageForText and searchForImageLogo and consolidates addresses and logos Lists into a MailResponse object |
| searchImageForText | Looks for the text in the image using Google Cloud. The method parameter is a base64 encoded string of the image. Vision will return a list of addresses found. |
| searchForImageLogo | Looks for logos on the image using Google Cloud vision and return a list of logos found. The method parameter is the string base64 encoded image. |

There are ten private methods used primarily by the searchImageToText method; see below for their name and a description of each.

**Table 4**

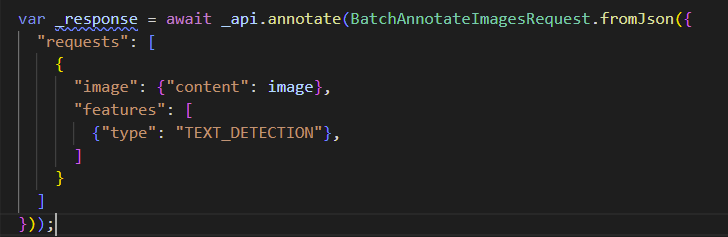
*Private methods for CloudVisionAPI class*

| **Method** | **Description** |
| --- | --- |
| \_parseBlockForAddress2 | The primary function used to parse response JSON from image processing.  It returns a List of Address objects. All are the below methods are used to help parse each block and extract addresses based on specific criteria |
| \_blockhasPostage | Validates whether a block contains a postage stamp |
| \_validateNameHasNoSpecialSymbols | Validates name has no special symbols |
| \_validateCityStateZip | Validate city, state, and zip based on the USPS guidelines |
| \_findLineWithCityStateZip | Find a line that contains city state zip within a block and returns its index position |
| \_validateAddress1 | Validates address follows mail guidelines |
| \_findLineWithAddress1 | Searches for the line that follows address1 guidelines and returns the line index within the current block. |
| \_findBlocksWithAddresses | Checks and determines which block contains zip and returns a list of potential blocks with addresses. |
| \_checkForUnits | Checks whether a line starts with a unit name/abbreviation. |
| \_findblocksWithAddresses1 | This function checks and determines which blocks contain an address line and returns a list of potential blocks with addresses. |

Once the CloudVisionAPI object has been instantiated, the application will use the search method and provide as a parameter the base64 encoded image to be input for both the OCR Image to Text and logo methods. When submitting the image processing request to Google cloud vision, the application submits the following request JSON body for Vision to process the image and waits for a response using the below await method.

**Figure 22**

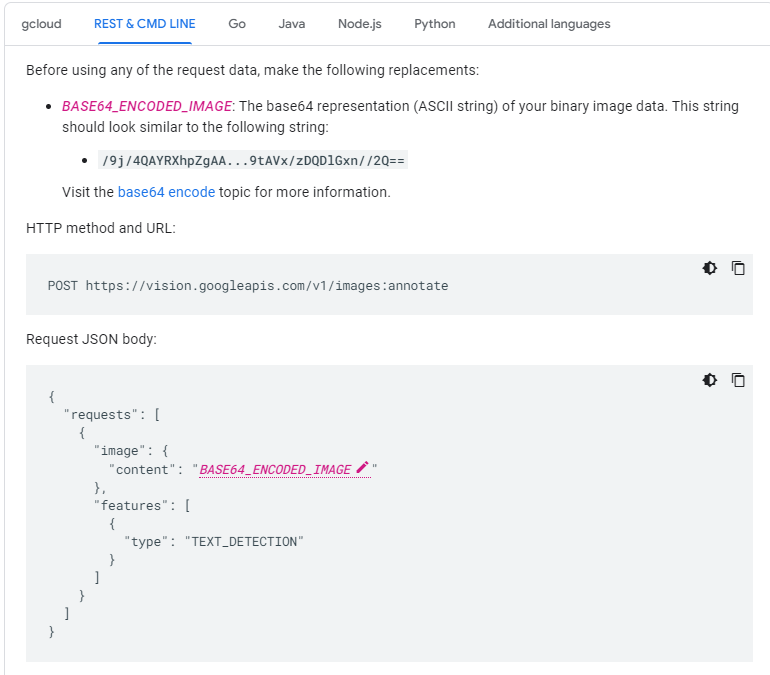
*SearchImageForText request function*



See below for a sample from the google\_vision\_api official tutorial.

**Figure 23**

*Google Cloud Vision Tutorial – OCR Image to test Request Body*



When the searchImageForText method receives a response, it consists of a TextAnnotation object. For the details, the application loops from granular to more refined in the table order listed below using the foreach method.

**Table 5**

*OCR Response Refinement Hierachy*

| **Vision OCR Response Refinement Process** |
| --- |
| Response |
| FullTextAnnotation |
| Page |
| Block |
| Paragraph |
| Words |
| symbol |

See below for its complete implementation.

**Figure 24**

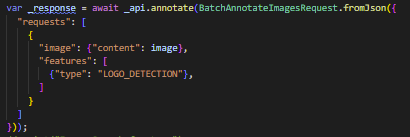
*Response parsing for block content*



Due to the spacing between different addresses, the application stored the content as blocks and extrapolated all the refined objects. The blocks were scanned for the city, state, and zip implementation based on USPS guidelines and determined which are potential addresses. The block index with the possible address is then sent to the parseBlockForAddress2 for analysis. The analysis returns a list of Address Objects. The Logo detection is very similar.

**Figure 25**

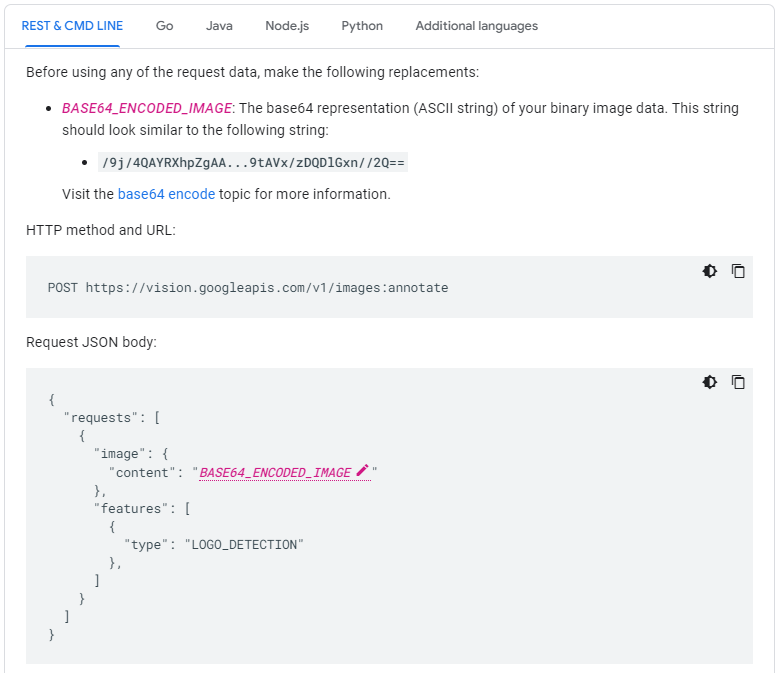
*Logo Request Code Implementation*

****

See below for a sample JSON request string from the Google Vision API official site.

**Figure 26**

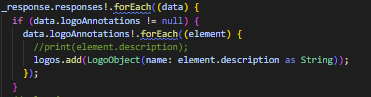
*Google Cloud Logo Sample Request*



The response was simple compared to OCR Image to Text, as it only had one loop to obtain the resulting logos.

**Figure 27**

*Logo Response Code Implementation*



7.3 Implementing USPS Address Validation

**Why use USPS Address Validation?**

USPS Address Validation is the functionality that allows the application to determine if the address that was found using Google Vision API is accurate. This is important so that the user can tell if the address is an actual address that they could respond to. This helps the user with understanding the content of the mail piece better.

**How does it work?**

The USPS Address Validation is a class that is meant to act as a middleman between the USPS Web API and the information from the Cloud Vision API. The operates with a static function that allows the application to pass a semicolon-delimited address string for the class to parse into an XML for USPS Web API. This allows for easy portability for different implementations. Handling the XML in this way allows the parser to only need XML Logic in a single place and keeps it off the Web API class in the instance that more would need to be added.

The USPS Web API class acts as an interface for the application to hand information too to successfully create the Web request to the API. This allows us to centralize all logic about the USPS Web API web interface portion into a single class. This means that if changes are ever needed to be made to the USPS Web API’s address, then it only needs to be changed in a single place.

**Classes and their Methods**

**USPS Address Verification**

As stated above this class is the area where the logic about parsing the address string takes place and where creating the XML is located.

**Table 6**

*USPS Address Parser Methods*

|  |  |
| --- | --- |
| **Method** | **Description** |
| Future<bool> verifyAddressString(String strAddr) async | This method is the only public function available on the class that is not a getter function. This is the function that will parse the address string based on the |
| String \_findZip5(String str) | This is the method that allows us to find the 5-digits of the zip code. This logic is powered using Regular Expressions and allows the logic for finding this part of the zip code in a single place. |
| String \_findZip4(String str) | This is the method that allows us to find the 4-digit extension of the 5-digit zip code if it is present. This logic is powered using Regular Expressions and allows the logic for finding this part of the zip code in a single place. |
| String \_findState(String str) | This is the method that allows us to find the state portion of the address. Using Regular Expressions we can determine if the state is two characters or the full state. Keeping this logic in a single place allows us to easily make changes to the logic as needed. |
| xml.XmlDocument \_buildAddressXml(String strAddr1, String strAddr2,String strCity, String strState, String strZip5, String strZip4) | This is the method that holds the logic for formatting the XML. The XML format is determined by the USPS Web API documentation that can be found [here](https://www.usps.com/business/web-tools-apis/address-information-api.pdf). We utilize the XML package here because it allows the code to remain cleaner and simpler to understand the structure of the XML. |

**USPS Web API**

As stated above this is the section of the USPS Address Validation that deals with directly interacting with the Web API. This is done so that as needs expand the class can be added to with new functionality.

**Table 7**

*USPS Web API Variables*

|  |  |
| --- | --- |
| **Variable** | **Description** |
| final String \_strUrl = "https://secure.shippingapis.com/"; | This is the base URL for the USPS Web API. Having it like this means that we are able to easily change and access this information as needed. |
| final String \_strUserID = "974UNIVE7445"; | This is the user id associated with this project that will allow the application to verify addresses against the API. Making this property like this makes it easy to replace when the project changes hands. |

**Table 8**

*USPS Web API Methods*

|  |  |
| --- | --- |
| **Method** | **Description** |
| Future<bool> verifyAddressXml(String xml) async | This is the method that should be called by any class that has created the address XML and wants to verify the address. This is the area in the code where the application will create the fully declared API request and then parse the returned XML for a response. This allows the logic that needs to follow USPS’ [documentation](https://www.usps.com/business/web-tools-apis/address-information-api.pdf) in a single place. |
| Future<bool> testConnectionToUspsAPI() async | This function exists to be run as a test to see if the USPS Web API is still online. This function is only used while testing because if requests fail then the application can not verify anyway. |
| Future<String> \_callClient(String strUri) async | This is the method that holds the logic for calling the HTTP client. The documentation for this can be found [here](https://pub.dev/packages/http). |

**How to Implement**

**USPS Address Verification**

To implement the class:

* Import the following package:

import '../lib/image\_processing/usps\_address\_verification.dart';

* Create the class object:

final uspsAddressValidator = UspsAddressVerification();

* Call the method to verify the address

Var result = await uspsAddressValidator.verifyAddressString(strAddress);

* The class will then return a boolean value based on if the address is valid.

**USPS Web API**

To implement the class:

* Import the following package:

import './usps\_web\_api.dart';

* Create an object of the class:

UspsWebApi webApi = UspsWebApi();

* Call the method to run the Web API Request

await webApi.verifyAddressXml(doc.outerXml);

* The method will return a boolean value based on the response of the request.

7.4 Implementing Digest Email Parsing/IMAP

**What is Digest Email Parsing?**

The Daily Digest Email Parsing is the functionality that implements the IMAP calls to retrieve a USPS Informed Delivery Daily Digest. To perform the IMAP functionality the class implements the enough\_mail package. This package allows us to use IMAP search commands to filter emails by a specific set of parameters which we can then use to create an IMAP fetch command to retrieve the desired email. This allows us to circumvent the need to have an archival system by using the user’s email as an archival system. But this is within the context that the user does not delete their digest emails.

**Why use IMAP?**

IMAP stands for Internet Message Access Protocol. These Protocols allow us to navigate a user’s email without having to see every email. Examining every email would be time-consuming and would not result in a positive user experience. So to provide the users with a positive experience IMAP is a simple solution to provide what is needed.

**How does it work?**

The class will create a custom object called Digest which will contain the MimeMessage of the email and then parsed information that is meant to be consumed by the UI. This allows us to easily access the information that is shown on the UI and allows us to more easily navigate the information from the UI.

To take advantage of IMAP both the search and fetch commands needed to be implemented properly to work together. By following the IMAP RFC3501 Documentation, found [here](https://datatracker.ietf.org/doc/html/rfc3501), the protocol criteria can be made specific enough that when making a search command we can obtain the sequence id of the email, and then using enough\_mail’s fetchMessage method we are able to call a fetch command that can quickly obtain the raw information of the email for the data to then be parsed out. This raw data is what is parsed out into the different Digest Models.

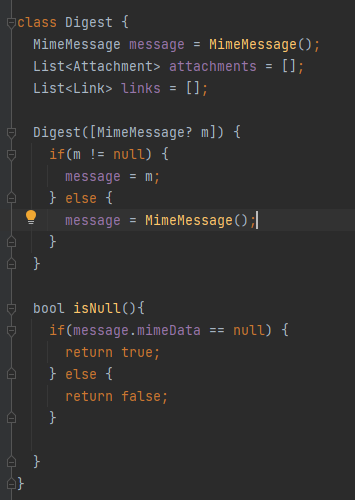
**Functions and Classes**

**Digest Models**

1. **Digest**

**Figure 28**

*Digest Class*

****

The Digest model is an all-encompassing breakdown of the relevant content of the USPS Daily Digest Email. The MimeMessage is the raw data that is returned by the IMAP protocol. The attachments are all the images built into their respective model. The links are all the hyperlinks on the email built into their respective model. Doing this allows to UI to more easily consume the data instead of having to directly interact with the MimeMessage.

**Table 9**

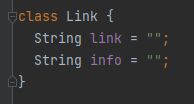
*Digest Class Methods and Description*

|  |  |
| --- | --- |
| **Method** | **Description** |
| bool isNull() | Returning a null to the UI when requesting a Digest that did not exist caused too many issues, it was decided to create a function that checked to see if the Digest was empty. This will check to see if the attached MimeMessage is null or not |

1. **Link**

**Figure 29**

*Link Class*

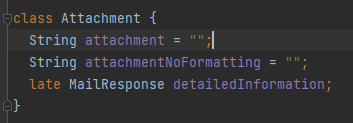


This Model is representative of the Hyperlinks that can be found on the email sent by USPS. This is important information as this can contain the details for packages and other links that can be found in the text of the email.

1. **Attachment**

**Figure 30**

*Attachment Class*



This model is representative of the Images attached to the Daily Digest Email. The attachment String attachment represents the base64 encoding of the image with the default formatting. This is used to display the image to the user on the UI without having to save the image locally. The attachmentNoFormatting is there because there are instances where having the default formatting will cause issues. The detailedInformation is the information retrieved from the Google Cloud Vision API. Having the MailResponse here allows us to associate the information with the respective image in a single object.

1. **Digest Parser**

**Table 10**

*Digest Parser Methods and Descriptions*

|  |  |
| --- | --- |
| **Method** | **Description** |
| Future<Digest> createDigest(String userName, String password, [DateTime? targetDate]) | This function will perform the needed functionality. targetDate can be passed optionally, if it is not passed then the class will supplement the current date as the targetDate. |
| Future<List<Attachment>> \_getAttachments(MimeMessage m) async | This method contains the logic to get the images attached to the email. This logic is separated out so that the method is easier to manage and presents a cleaner code. This method also works into the logic for the Google Vision API. |
| List<Link> \_getLinks(MimeMessage m) | This method is the logic for collecting the links on the email. This method uses Regular Expressions to find all hyperlinks and deconstructs the text will all are extracted. This is done so that the entire email is covered, |
| String \_formatTargetDateForSearch(DateTime date) | This method exists to format dates so that they can be used in the IMAP Search command’s filter criteria. |
| Future<MimeMessage> \_getDigestEmail() async | This is the method that implements the Logic to retrieve the desired Daily Digest Email. This is where the IMAP client is created and the Digest’s MimeMessage is returned. |
| deleteImageFiles() async | The is the method that removes any images saved to the device by the application. This is done to ensure that the images being sent to the QR scanner are the images associated with the current digest. |
| Future<bool> saveImageFile(Uint8List imageBytes, String fileName) async | This is the method that saves the images to a specific location on the device. This is done because the QR/Barcode logic needs the image stored locally to interpret information. |
| Future<bool> \_requestPermission(Permission permission) async | This is the method that requests the user’s permission to access their local images. This is needed so that we can save the images locally. |
| Future<MailResponse> processImage(String imagePath) async | This is the method that gets the Google Vision API information, including the QR/Barcode information. In this method, we pass the path of the current image attachment that was saved locally to the device. |

**How to Implement**

To implement the class:

* Import the following package

import 'package:summer2022/digest\_email\_parser.dart';

* From there the function can be called by passing the username and password of the account the user signed in with.

await DigestEmailParser().createDigest(await Keychain().getUsername(), await Keychain().getPassword());

* + The function’s option parameter can be passed in like this:

await DigestEmailParser().createDigest(await Keychain().getUsername(), await Keychain().getPassword(), selectedDate);

* 1. Implementing Google ML Kit

Google ML Kit is a machine learning package developed by Google, optimized for iOS and Android applications. This package has several vision and natural language processing features, such as barcode scanning and language detection. Additionally, Google ML Kit is free of cost to use and has an active developer community on GitHub and StackOverflow in case any issues are detected. As one of the use cases for the USPS Informed Delivery App was to detect barcodes and return their associated data, Google ML Kit’s Barcode Scanning feature was considered as an option to perform this function. After comparing the Google ML Kit Barcode Scanning feature to comparable Flutter plugins, such as flutter\_barcode\_scanner, it was concluded that the Google ML Kit plugin for Flutter had more thorough documentation for developers’ reference, a popularity score of 97% from users on pub.dev, and had successfully detected 76% of barcodes in a test of 100 randomized mailpieces with barcodes.

Before implementing the Google ML Kit Barcode Scanning feature, it is assumed that the Flutter plugin for the library has been installed and is visible in the pubspec.yaml file of the application. If not, please follow the install guide: <https://pub.dev/packages/google_mlkit_barcode_scanning/install>

**Figure 31**

*Barcode Scanning plugin in pubspec.yaml*

Text

Description automatically generated

If the plugin has been installed successfully, it can be used in the Dart code by importing:

import 'package:google\_mlkit\_barcode\_scanning/google\_mlkit\_barcode\_scanning.dart';

at the start of the Dart file before any classes are declared. An object of instance BarcodeScanner will be used to parse images for barcodes. BarcodeScanner objects only take image objects of type InputImage.

To create a BarcodeScanner object:

final BarcodeScanner \_barcodeScanner = BarcodeScanner();

There are various methods to create an InputImage object using an existing image file: by filepath, by bytes, or from a File object.

**Table 11**

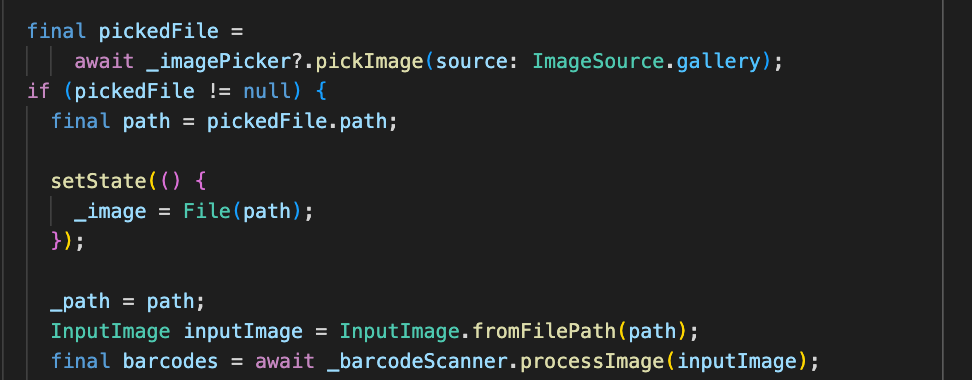
*Methods of creating an InputImage*

|  |  |
| --- | --- |
| **Method** | **Description** |
| InputImage.fromFilePath(path) | Takes a string value of the image file’s relative or absolute file path. |
| InputImage.fromFile(img) | The image is passed as an object of type File. |
| InputImage.fromBytes(bytes: bytes, inputImageData: inputImageData) | The image is converted to and passed as an byte array, along with its input image data (such as orientation and bounding). |

Once an InputImage object is created for the chosen image, the value can be passed into the BarcodeScanner object’s processImage() method.

**Figure 32**

*Example of passing an InputImage into processImage()*



The processImage method outputs a list of type Barcode. Each Barcode contains pertinent attributes, such as the barcode’s type and value.

An example of a Barcode object’s attributes for a QR Code:

{

"type": 8,

"format": 10,

"displayValue": "https://qrco.de/bczuEB",

"rawValue": "https://qrco.de/bczuEB",

"rawBytes": Uint8List.fromList([

104,

116,

116,

112,

115,

58,

47,

47,

113,

114,

99,

111,

46,

100,

101,

47,

98,

99,

122,

117,

69,

66

]),

"boundingBoxLeft": 394.0,

"boundingBoxTop": 619.0,

"boundingBoxRight": 505.0,

"boundingBoxBottom": 730.0,

"cornerPoints": [

{"x": 394, "y": 619},

{"x": 505, "y": 619},

{"x": 504, "y": 727},

{"x": 395, "y": 730}

],

"url": "https://qrco.de/bczuEB",

"title": "TD Bank"

}

In this json, the barcode type and format are index values that correspond to a BarcodeType object and BarcodeFormat object in their respective arrays. The BarcodeType in this case is Barcode.TYPE\_URL, while the BarcodeFormat is Barcode.FORMAT\_QR\_CODE.

A list of all BarcodeFormat types: <https://developers.google.com/android/reference/com/google/mlkit/vision/barcode/common/Barcode.BarcodeFormat>

A list of all BarcodeType types (please note the difference in class names): <https://developers.google.com/android/reference/com/google/mlkit/vision/barcode/common/Barcode.BarcodeValueType>

This application utilizes the type and rawValue attributes for each barcode.

**Figure 33**

*Calling the rawValue attribute*

Text

Description automatically generated

# Testing

8.1 Create a unit test

We use the default testing package because that is the most easily portable across all systems and saves the team time from having to deal with any versioning issues. This allows us to keep the unit tests simple and their implementation easily expandable.

1. Add the test or flutter\_test dependency.

**Figure 34**

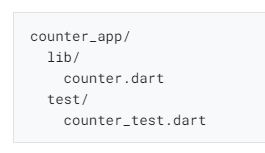
*Flutter Test Dependency*



1. Create a test file.

**Figure 35**

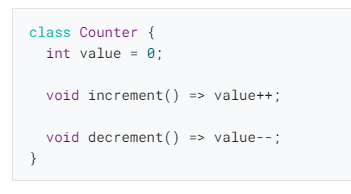
*Creating Test File*



1. Create a class to test.

**Figure 36**

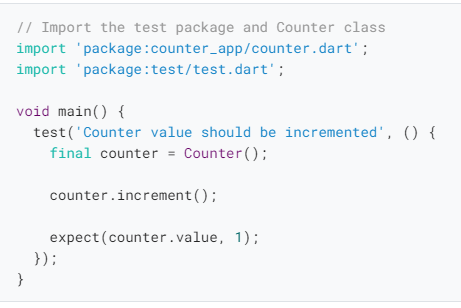
*Test Class*



1. Write a test for our class.

**Figure 37**

*Counter Test*



1. Combine multiple tests in a group.

**Figure 38**

*Creating a Test Group*



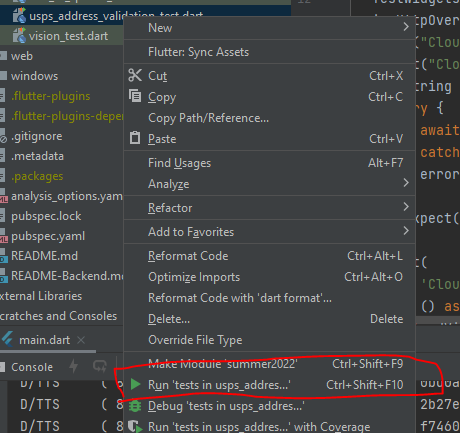
* 1. Execute Unit Test

The Flutter plugins for IntelliJ and VSCode support running tests. This is often the best option while writing tests because it provides the fastest feedback loop as well as the ability to set breakpoints.

1. IntelliJ
   1. Open the counter\_test.dart file
   2. Select the Run menu
   3. Click the Run 'tests in counter\_test.dart' option
   4. Alternatively, use the appropriate keyboard shortcut for your platform.
2. VSCode
   1. Open the counter\_test.dart file
   2. Select the Run menu
   3. Click the Start Debugging option
   4. Alternatively, use the appropriate keyboard shortcut for your platform.
3. Android Studio
   1. Right-click the test file and select run tests.

**Figure 39**

*How to Run Tests*



# Appendices

9.1 References

*An introduction to unit testing* (n.d.) Flutter. Retrieved July 26, 2022, from <https://docs.flutter.dev/cookbook/testing/unit/introduction>

*Flutter - Build apps for any screen*. (n.d.). Flutter. Retrieved July 26, 2022, from

<https://flutter.dev/>

*Fast Facts of Common Eye Disorders | CDC*. (n.d.). CDC. Retrieved May 31, 2022, from<https://www.cdc.gov/visionhealth/basics/ced/fastfacts.htm#:%7E:text=Approximately%2012%20million%20people%2040,due%20to%20uncorrected%20refractive%20error>.

*GitHub Desktop Tutorial – Collaborate With GitHub From Your Desktop* (July 16, 2022) SoftwareTestingHelp. Retrieved July 26, 2022, from <https://www.softwaretestinghelp.com/github-desktop-tutorial/>

*Run app on a hardware device* (n.d.) Andriod. Retrieved July 26, 2022, from <https://developer.android.com/studio/run/device>

*Windows install*. (n.d.). Flutter. Retrieved July 26, 2022, from <https://docs.flutter.dev/get-started/install/windows>

*iOS install.* (n.d). Flutter. Retrieved July 26, 2022 from

[macOS install | Flutter](https://docs.flutter.dev/get-started/install/macos)

*Linux install.* (n.d). Flutter. Retrieved July 26, 2022 from

[Linux install | Flutter](https://docs.flutter.dev/get-started/install/linux)

*Chrome OS install.* (n.d) Flutter. Retrieved July 26, 2022 from

<https://docs.flutter.dev/get-started/install/chromeos>

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