

Programmer's Guide

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-Programmer's Guide |
| Document Owner | UMGC SWEN 670 TEAM A |
| Issue Date | October 29, 2022 |
| Last Saved Date | October 29, 2022 |
| File Name | USPS\_Informed\_Delivery\_App\_Enhancements\_PG\_Team\_A.docx |

Document History

|  |  |  |
| --- | --- | --- |
| Version | Issue Date | Changes |
| 1.0 | 10/29/2022 | Initial Version |
| 1.1 | 11/05/2022 | Updated captions and provided sources for online images. |
|  |  |  |
|  |  |  |

Contents

[1 Introduction 1](#_Toc118042787)

[1.1 Purpose 1](#_Toc118042788)

[1.2 Intended Audience 1](#_Toc118042789)

[1.3 Project Documents 1](#_Toc118042790)

[1.4 Technical Project Stakeholders 1](#_Toc118042791)

[1.5 Acronyms and Abbreviations 2](#_Toc118042792)

[1.6 References 3](#_Toc118042793)

[2 Operating Systems and Development Tool Versions 5](#_Toc118042794)

[3 Development Environment 6](#_Toc118042795)

[3.1 Integrated Development Environments (IDE) 6](#_Toc118042796)

[3.1.1 Android Studio 6](#_Toc118042797)

[3.2 Emulators and Simulators 8](#_Toc118042798)

[3.2.1 Android Emulator 8](#_Toc118042799)

[3.2.2 Android Device (Physical) 9](#_Toc118042800)

[3.2.3 iOS Simulator 10](#_Toc118042801)

[3.2.4 iOS Device 11](#_Toc118042802)

[4 Frameworks and Languages 12](#_Toc118042803)

[4.1 Flutter 12](#_Toc118042804)

[4.1.1 Flutter Doctor 13](#_Toc118042805)

[4.1.2 Hot Reload 13](#_Toc118042806)

[4.1.3 Flutter Widget Inspector 14](#_Toc118042807)

[4.1.4 Installation 14](#_Toc118042808)

[4.2 Dart 19](#_Toc118042809)

[4.2.1 Null Safety 19](#_Toc118042810)

[4.2.2 Installation 19](#_Toc118042811)

[5 Development Process 21](#_Toc118042812)

[5.1 GitHub 21](#_Toc118042813)

[5.1.1 Cloning 22](#_Toc118042814)

[5.1.2 Branching 23](#_Toc118042815)

[5.1.3 Merging 25](#_Toc118042816)

[5.1.4 Pull Request 25](#_Toc118042817)

[6 Code Structure 27](#_Toc118042818)

[6.1 State Management 27](#_Toc118042819)

[6.2 Asynchronous operation 27](#_Toc118042820)

[6.3 Android 28](#_Toc118042821)

[6.4 iOS 28](#_Toc118042822)

[6.5 Common code functions 28](#_Toc118042823)

[6.6 Keychain 28](#_Toc118042824)

[6.7 Implementing Google Cloud Vision Application Programming Interface (API) 29](#_Toc118042825)

[6.8 Implementing Digest Email Parsing/Internet Mail Application Protocol (IMAP) 30](#_Toc118042826)

[6.9 Implementing Mail View 31](#_Toc118042827)

[6.9.1 Mail Actions 31](#_Toc118042828)

[6.9.2 Email addresses and phone numbers 31](#_Toc118042829)

[6.9.3 Sender Detection 32](#_Toc118042830)

[6.9.4 Displaying Individual Mail Pieces 33](#_Toc118042831)

[6.9.5 Email Services 35](#_Toc118042832)

[6.10 Implementing Notifications 35](#_Toc118042833)

[6.10.1 Notifications View 36](#_Toc118042834)

[6.10.2 Create Notifications Subscription 36](#_Toc118042835)

[6.10.3 Delete Notifications Subscription 37](#_Toc118042836)

[6.10.4 Clear Notification 37](#_Toc118042837)

[6.10.5 Notification Management View 37](#_Toc118042838)

[6.10.6 Notification fetch process 37](#_Toc118042839)

[6.10.7 Navigate to mail item from notification 38](#_Toc118042840)

[6.11 Digest Modification 38](#_Toc118042841)

[6.11.1 Digest Modifications 38](#_Toc118042842)

[6.12 Implementation of Voice Assistants 39](#_Toc118042843)

[6.12.1 Android Assistant Integration 39](#_Toc118042844)

[7 Project File Structure 40](#_Toc118042845)

[7.1 Overview 41](#_Toc118042846)

[7.2 Pubspec.yaml 42](#_Toc118042847)

[7.2.1 Description 42](#_Toc118042848)

[7.2.2 Dependencies 42](#_Toc118042849)

[7.2.3 Assets 45](#_Toc118042850)

[8 User Interfaces (UI) 46](#_Toc118042851)

[8.1 Front-End Screens 46](#_Toc118042852)

[9 Accessibility 60](#_Toc118042853)

[9.1 Gestures 61](#_Toc118042854)

[9.2 Semantics 62](#_Toc118042855)

[10 Data and Backend 63](#_Toc118042856)

[10.1 SQLite Database 64](#_Toc118042857)

[10.2 Notification and Notification Subscription Data Object – GitHub #412 65](#_Toc118042858)

[11 Firebase Analytics 66](#_Toc118042859)

[11.1 Installation using Firebase CLI and Flutterfire 66](#_Toc118042860)

[11.2 Setup / Configuration of Analytics Service 67](#_Toc118042861)

[11.3 Firebase Debug View 69](#_Toc118042862)

[12 Publishing 71](#_Toc118042863)

[12.1 Google Play 71](#_Toc118042864)

[12.2 Apple Appstore 72](#_Toc118042865)

[13 Appendix 74](#_Toc118042866)

[13.1 Credits 74](#_Toc118042867)

[13.2 Credit to Previous Cohorts 75](#_Toc118042868)

# Introduction

## Purpose

The MailSpeak App Programmer's Guide is a document that takes the audience through tools with setup, showing the app architecture, the repository handling, the development, and explaining the choices made in general. It is a good reference document for other programmers explaining the why and how of the development process and whether to only understand it or continue working on it.

## Intended Audience

The primary audiences of this document are the project stakeholders and likely future developers who would continue to work on the software, as it is a good reference document in both technical and informative form to understand the work done and the environment within.

## Project Documents

The following documents are included in the project's software 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 |

Table Project documents

## Technical Project Stakeholders

The project stakeholders for the MailSpeak App – United States Postal Service (USPS) Informed Delivery Enhancement development project are shown in the table below

|  |  |
| --- | --- |
| Name | Role |
| Dr. Mir Assadullah | Professor |
| Roy Gordon | Project Mentor |
| Robert Wilson | Project Mentor |
| Mike Conatser | Overall project Manager + |
| Christopher Thorn | Product Owner + |
| Jack Shira | (SE III) Senior Developer + |
| Imoh Noah Jr | (SE III) Senior Developer + |
| Ali Fahimi | (SE II) Senior Developer + |
| Harsh Gadani | (SE II) Developer + |
| Jahan Brahamabhatt | (SE II) Developer + |
| Natan Tafese | (SE I) Developer + |
| Fahed Masood | (SE I) Developer + |
| Scott Huber | (SE I) Developer + |
| Kuleni Digga | (SE I) Developer + |

Table Project stakeholders

## Acronyms and Abbreviations

|  |  |
| --- | --- |
| AES | Advanced Encryption Standard |
| API - | Application Program Interface |
| BA - | Business Analyst |
| CE - | Code Editor |
| GCP - | Google Cloud Service Platform |
| HI - | Hardware Interface |
| IDE - | Integrated Development Environment |
| iOS - | iPhone Operating System |
| OS | Operating System |
| PM - | Project Manager |
| RSA | Rivest-Shamir-Adleman Algorithm |
| SDK - | Software Development Kit |
| SE - | Software Engineer |
| SI - | Software Interface |
| UI - | User Interface |
| UMGC- | University of Maryland Global Campus |
| UML - | Unified Modeling Language |
| VM - | Virtual Machine |

## References

*Asynchronous programming: futures, async, await.* (n.d.). Retrieved October 28, 2022, from <https://dart.dev/codelabs/async-await>

Boluwatife. (June 4, 2022). *Using Regular Expressions (RegExp) in Dart/Flutter.* Retrieved October 28, 2022 from <https://dev.to/0xba1/using-regular-expressionsregexp-in-dartflutter-3p1j>

*Dart packages*. (n.d.). Dart packages. Retrieved October 8, 2022, from <https://pub.dev/>

De Jesus, S. Detoito, A. Gartrell, B. Johnpeter, S. Sampson, R. Srinivasan, A. (n.d). *Programmer Guide USPS Informed Delivery App – Visually Impaired.*

flutter-dev@. (n.d.). *Flutter and the pubspec* Retrieved October 7, 2022, from <https://docs.flutter.dev/development/tools/pubspec>

flutter-dev@. *Using packages*. Flutter. (n.d.). Retrieved October 7, 2022, from <https://docs.flutter.dev/development/packages-and-plugins/using-packages>

*Get the dart SDK*. Dart. (n.d.). Retrieved October 2, 2022, from <https://dart.dev/get-dart>

*Keychain Services*. Apple (n.d.). Retrieved October 24, 2022 from  
<https://developer.apple.com/documentation/security/keychain_services?ref=morioh.com&utm_source=morioh.com#//apple_ref/doc/uid/TP30000897-CH203-TP1>

Mohite, J. (2020, Dec, 13) Retrieved October 9, 2022 from <https://medium.com/flutterworld/flutter-mvvm-architecture-f8bed2521958>

Nair, S. (2020, Aug, 24) *GitHub for Data Scientists: Part 2 | by Snehal Nair | Towards Data Science Retrieved November 5, 2022 from GitHub for Data Scientists: Part 2 | by Snehal Nair | Towards Data Science* from https://towardsdatascience.com/collaborate-on-github-like-pro-part2-26a6790e8f93

*Navigation and Routing*. (n.d.). Retrieved October 29, 2022 from <https://docs.flutter.dev/development/ui/navigation>

Ori, E. (2021, May 26). Mastering Flutter: Semantics, from [Mastering Flutter: Semantics. Accessibility is a hard subject, as it… | by Enrico Ori | TheOtherDev/s | Medium](https://medium.com/theotherdev-s/mastering-flutter-semantics-672440bc8bc8)

*Run apps on the Android emulator: Android developers*. Android Developers. (n.d.). Retrieved October 2, 2022, from <https://developer.android.com/studio/run/emulator>

SAMSUNG. (2022, September 21). *How do I turn on the developer Options Menu on My Samsung Galaxy Device?* Samsung uk. Retrieved October 5, 2022, from <https://www.samsung.com/uk/support/mobile-devices/how-do-i-turn-on-the-developer-options-menu-on-my-samsung-galaxy-device/>

# Operating Systems and Development Tool Versions

The following table contains a list of all versions of software used during this development project.

|  |  |
| --- | --- |
| Product Name | Version |
| Windows 10 | OS Build 19041.1415 |
| Android Studio Dolphin | 2021.3.1 Patch 1 |
| GitHub Desktop | Version 3.1.2 (x64) |
| Flutter | Version 3.3.6 |
| Dart |  |
| Git | 2.38.1 |

Table Project Software Tool Versions

# Development Environment

## Integrated Development Environments (IDE)

### Android Studio

Android Studio is an Integrated Development Environment (IDE) for Android development and includes everything one needs to build android apps. It is an IDE specifically designed for developing Android applications. As Flutter is a framework created for multiplatform application development, the Android Studio IDE is utilized for developing native Flutter applications for Android.

#### Download and Installation

**Windows**

*Environment Specifications:*

* Windows 10 or later (64-bit), x86-64-based,
* 8 GB disk space,
* 8 GB random access memory (RAM)

**Package:**

* [EXE Installer (Preferred)](https://redirector.gvt1.com/edgedl/android/studio/install/2021.3.1.17/android-studio-2021.3.1.17-windows.exe)
* [ZIP Installation](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 the machine

1. Follow the setup wizard and install the software development kit (SDK) dependencies

**macOS**

*Environment Specifications:*

* macOS Mojave or higher,
* 8 GB disk space,
* 8 GB random access memory (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 disk image (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 Installer](https://developer.android.com/studio#downloads:~:text=android%2Dstudio%2D2021.2.1.15%2Dlinux.tar.gz)

**Installation:**

1. Extract the tar.gz file and move the 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

**Chrome Operating System (OS)**

*Environment Specifications:*

* Chrome OS (64-bit),
* 4 GB disk space,
* 8 GB RAM,
* Linux for Chrome enabled

**Package:**

* [Chrome Installer](https://developer.android.com/studio#downloads:~:text=android%2Dstudio%2D2021.2.1.15%2Dcros.deb)

**Installation:**

1. Open the Files app and locate the Debian (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

## Emulators and Simulators

### Android Emulator

An Android Emulator, included with Android Studio, enables testing the app on different android devices without needing to own a physical android device. The emulator installs and starts the app faster. The computer specs to use the emulator in Android Studio smoothly are:16GB RAM, 64-bit Windows, macOS, or Linux operating system, and 16GB disk space.

The workflow for using the emulator:

1. Check your Specs.
2. Create a Virtual Device:
3. Run the app on the emulator
4. Navigate the emulator

Run App on Android Emulator: <https://developer.android.com/studio/run/emulator>

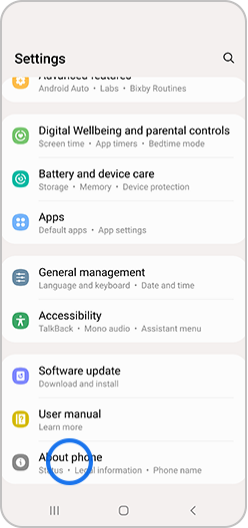
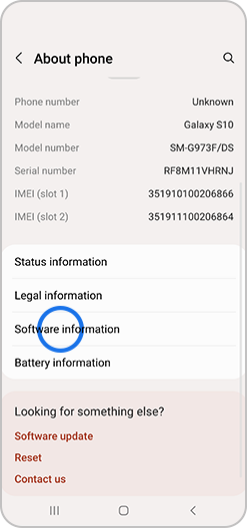
### Android Device (Physical)

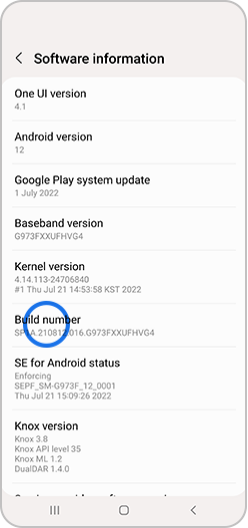
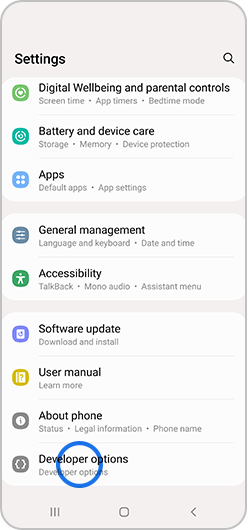
It's important to always test the app on an actual device before releasing it to users. Furthermore, using the physical Android device should not hinder using the Android emulator in the same project. The former ensures the product for release, while the latter tests the app's functionality on different versions of the Android platform.

**Before starting Debug:**

1. Decide the connection type, Wi-Fi or Universal Serial Bus (USB), to connect with the device

2. On the device, open the **Settings** app, select **Developer options**, and then enable **USB**  **debugging** (if applicable)

**Pattern/pin****Developer** option appears Under setting

3. Set up your system to detect your device

### iOS Simulator

In XCode, open your project's workspace. At the top of the XCode application, directly in the center, is a section selecting a simulator device. Click on this tab as seen in the photo:

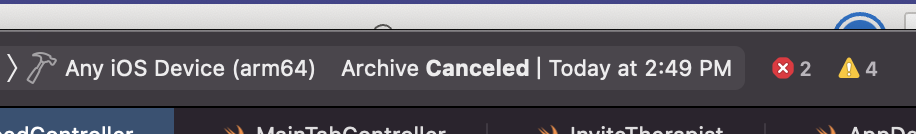


Figure 1 iOS Simulator tab

After clicking this, a drop-down will appear with several simulators of iPhones and iPads. 

Figure 2 iOS simulator options

Select the simulator you would like to test on. Once the simulator is selected, hit the play/run button on the application, and the simulator will launch using your Macs hardware to run your application. Then you can test your application as needed.

### iOS Device

The first step to running a physical Apple device is to plug your iPhone or iPad into your Mac with a USB or lighting cable depending on your Mac. Next, XCode will ask if you trust this device, select 'Trust.'

After XCode recognizes your device, follow the steps above, and you should see your device in that drop-down in the picture from 2.2.3. Next, select your device and hit play/run. The app will then be built on your physical device, and you can begin testing.

# Frameworks and Languages

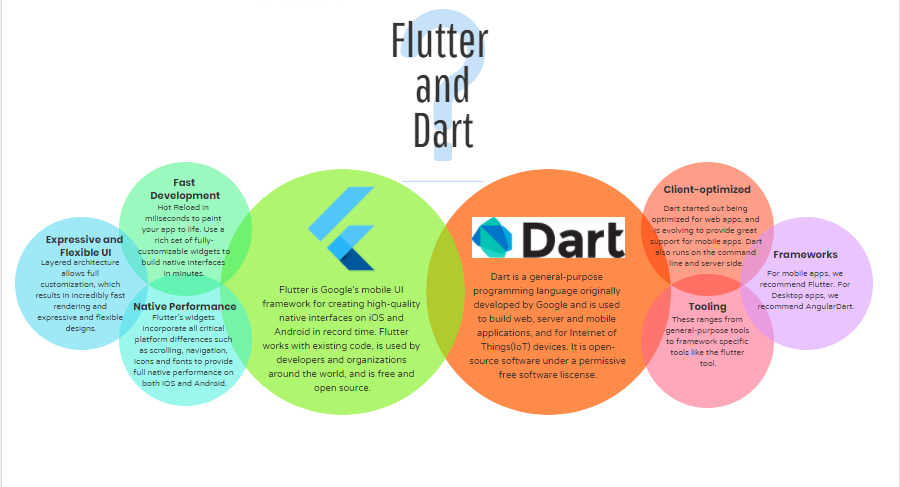


Figure Flutter and Dart Framework –

## From: https://www.geeksforgeeks.org/flutter-an-introduction-to-the-open-source-sdk-by-google/Flutter

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 an application off a single codebase.

While the Dart SDK can be installed separately, the Dart SDK with command line interface (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>

### Flutter Doctor

Flutter Doctor is a tool that checks the environment to make sure that it is setup optimally for both iOS and Android projects. It inspects the local development environment to make sure that all the necessary tools have been installed correctly and are configured correctly so that the framework is able to access them during build process. If errors are found, flutter doctor will provide hints on what issues need to be addressed and how to fix them.

Text

Description automatically generated

Figure - Flutter Doctor

### Hot Reload

Being able to visualize code changes immediately is a great advantage of Flutter framework. Hot Reload is an advancement of Flutter when combined with the Dart VM. It works by providing a route that allows the IDE to push code changes directly into a running version of the application if it is running in debug mode. Advancements in Hot Reload have increased refresh times as much as 30% for larger applications.

Hot Reload can be accessed by choosing the following icon on the console in Android Studio.  or typing r and pressing Enter in the terminal.

### Flutter Widget Inspector

Flutter Widget Inspector is a very useful tool for troubleshooting application rending issues. This tool is able to identify problems that occur with the visual layout. For more information, please refer to the official documentation at the following location: <https://docs.flutter.dev/development/tools/devtools/inspector>.

A screenshot of a computer

Description automatically generated

Figure 5 Flutter Inspector

### Installation

#### Windows

*Environment Specifications:*

* Windows 10 or later (64-bit), x86-64 based,
* 1.64 GB disk space,
* Git for Windows 2.x
* Windows PowerShell 5.0 or newer 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 (Do not install Flutter to a path that contains special characters or spaces. Also do not install Flutter in a directory like C:\Program Files\ that requires elevated privileges)
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.

Graphical user interface, application

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

Figure 7 Edit Windows Path Variables

Figure 6 Windows Environment Variables

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

Text

Description automatically generated

Figure 8 Flutter Doctor

#### macOS

*Environment Specifications:*

* macOS,
* 2.8 Gigabyte (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

Text

Description automatically generated

Figure 9 Flutter Doctor

#### 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

Text

Description automatically generated

Figure 10 Flutter Doctor

#### Chrome OS

*Environment Specifications:*

* Chrome OS (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

Text

Description automatically generated

Figure 11 Flutter Doctor

## Dart

Dart is a programming language for multiplatform development of web and mobile applications. The programming language can be used to build server and desktop applications. This project works on an iOS and Android application using Dart with the Flutter framework, both free open-source resources of Google.

The Dart SDK has the libraries and command-line tools to develop Dart command-line, server, and non-Flutter web apps. One can install Dart SDK separately from <https://dart.dev/get-dart> , or the Flutter SDK includes the full Dart SDK, and <https://dart.dev/tools/dart-tool> has Dart's command-line interface in its bin folder. (For this document it's the full Dart SDK included in Flutter SDK in use)

### Null Safety

The Dart language include sound null safety protection. With sound null safety, variables are never 'non-nullable' by default meaning that they can only be assigned values of a type that is not a null value. For more information, please refer to the following page: <https://dart.dev/null-safety>.

### Installation

Although the full Dart SDK is now included with Flutter, binaries may also be downloaded from the following location. As indicated in the image below, there are binaries for macOS, Linux, and Windows. <https://dart.dev/get-dart/archive#download-urls>

Graphical user interface, application

Description automatically generated

Figure 12 Dart Binaries Download

# Development Process

The repository is located on GitHub at https://github.com/umgc/fall2022.git. It can be accessed via the command line or through GitHub Desktop (shown below), 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.

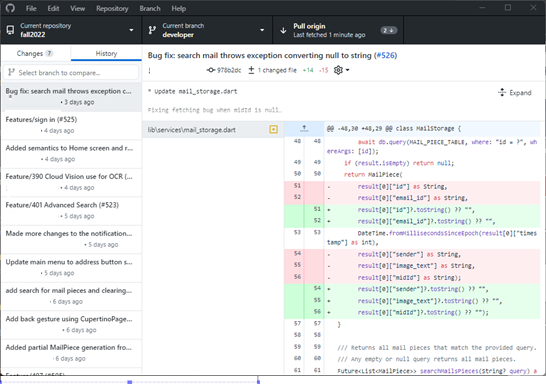


Figure 13 GitHub Desktop

## GitHub

GitHub is the online code repository that can be accessed using a variety of methods which include command line and GUI tools. This is the chosen location of the Fall 2022 code repository.

**Developer (default branch)** : git@github.com:umgc/fall2022.git

**Stats for Fall2022 Cohort:** **77** branches created **121** Closed Pull Requests

### Cloning

The team's repository is located on GitHub at [umgc/fall2022: SWEN 670 Fall 2022 cohort (github.com)](https://github.com/umgc/fall2022) Navigating to GitHub.com, you will have the option to download a zip archive or clone the repository as shown by the following image. If you only need to access the files for a short time, then downloading the zip file may make the most sense. However, in a development scenario when you'll need to consistently download updated files, cloning a repository will allow you to maintain an updated copy locally, while also pushing updates back to the server.

To clone the repository, ensure you have Git installed on your computer, and copy the link provided. Open a Git Bash or Git Terminal window and enter git clone <enter paste in the URL provided from GitHub>.

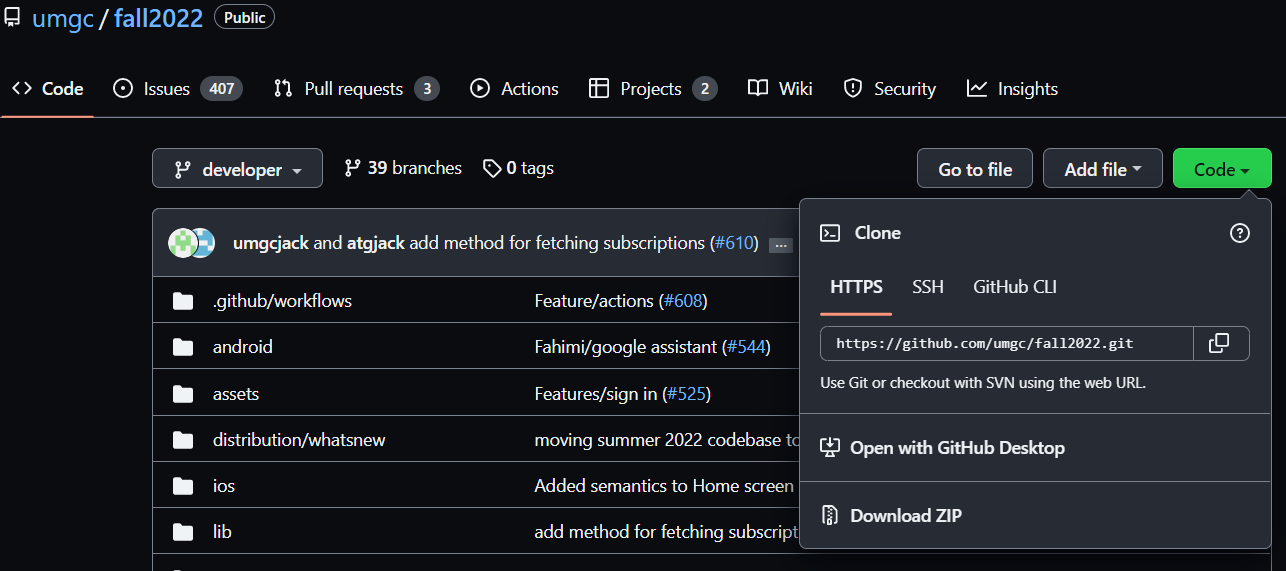


Figure 14 GitHub Project View

On your command prompt, use the command below

Text

Description automatically generated

Figure 15 Cloning Project

### Branching

The use of branching is to isolate development work without affecting other branches in the repository. Branching will let the developer safely experiment with new ideas, fix bugs, or develop features isolated in a contained area. One must have a write access to a repository to push any change in the form of a commit to push or upload their branch back to the master repository.

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

Application

Description automatically generated with medium confidence

Figure 16 - Github Flow

from https://towardsdatascience.com/collaborate-on-github-like-pro-part2-26a6790e8f93

The GitHub Branches breakdown has a Master, developer, and Feature. Master is a default name used when first branch is created. All repository has one since the git init command creates it by default. The branch contains the production code, and all development code is merged into master in sometime. It requires the DevSecOps and Project Manager (PM) approval before updating this branch.

Committing is the process that records changes in the repository as a snapshot of the status of the project. Pushing sends the recent commit history from the local repository up to GitHub.

Graphical user interface, application

Description automatically generated

Figure GitHub Interface

A Developer branch is created from the master branch. It is where all changes made by developers are held before merging back to the master for release, whereas a Feature branch is a supporting branch created from the developer. It is where a new feature is coded and tested by a team member until it is complete and ready for integration. Once satisfied with the work, a pull request is opened to merge the changes in the feature branch into developer.

Diagram

Description automatically generated with medium confidence

Figure Git Tree Diagram

To push the developer branch into the main branch, GitHub will perform a set of automated integration test to ensure the code that is integrated without negatively impacting the stability of the entire system. Once those tests complete successfully, two manual approvals are required by team members. This branch is used for deploying the application to the App Stores supporting Android and iOS.

### Merging

Once the work on the branch is complete, the next step is to merge the feature branch or bug fix branch into a master branch. Merging takes the changes done on the branch and implements incorporates them into the parent branch.

Steps to Merge Branches in GitHub Desktop

* In GitHub Desktop, click **Current Branch**
* Click Choose a branch **developer**
* Click the **feature**branch to merge to developer then click Merge **Branch into developer**. (If there is a merge conflict, GitHub will warn the user)
* Click **Push origin** to push your local changes to the remote repository.

Graphical user interface, application

Description automatically generated

A picture containing graphical user interface

Description automatically generated

Figure - Choosing Branch To Merge

### Pull Request

A pull request is the act of notifying the team that you have completed a set of changes and are requesting to incorporate those changes into the master (developer) repository. This process requires that you have already pushed a feature branch to the repository and are now requesting the additional review and sign-off of at least two team members to requesting the target repository to grab changes. In GitHub, once the branch has been pushed. Click on create pull requests and select the team members to inform them about changes pushed to a branch in the repository. Once a pull request is opened, changes are discussed and reviewed by team members/developers, and add follow-up commits before merging into the developer branch.

**Creating a pull request**

* Switch to the branch that you want to create a pull request for.
* Click Create Pull Request.
* On GitHub, confirm that the branch in the base: the drop-down menu is the branch where you want to merge your changes.
* Type a title and description for your pull request.

Graphical user interface, text, application, email

Description automatically generated

Figure - Creating Pull Request (PR)

# Code Structure

The MailSpeak app was developed using the Flutter framework and the Dart language. The code structure for the MailSpeak 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 broken down based on features. The following image illustrates the file structure in code.

## State Management

There are various state management schemes, and one that was considered is the BLOC (short for business logic) method where business logic is stored in a business logic layer. It was decided to use stateful widgets since the BLOC method was more complex to implement and not already used by the prior summer development team.

Stateless Widgets do not change state and only need to be rendered once. Stateful Widgets can change state. They are defined with an initial state and then can redraw the user interface based on the setState function. Some of the user interface pages require this ability to render the screen after obtaining alternate data at a later time.

## Asynchronous operation

Dart is a single thread or process loop programming language, so there are features that help it process lengthy operations while not delaying other computations. These are called asynchronous operations. Future is a datatype of typical data types that returns a value later after the operation is finished. It is typically used along with the async keyword future function definition. Await is a keyword telling the function to wait for an async function response, which is used for operations dependent on the output of a specific asynchronous function (*Asynchronous programming*, n.d.).

Asynchronous operations are used in various places in the application that deal with obtaining information from a network such as email fetching.

## Android

This folder is maintained by the Flutter framework and contains the files necessary to build the android specific installer.

## iOS

This folder is maintained by the Flutter framework and contains the files necessary to build the iOS specific installer.

## Common code functions

RegExp: Short for “Regular Expression”. Finding specific groups of text such as links or html text is done with the function RegExp. This is a tool for searching for specific types of strings (Boluwatife, June 4, 2022).

Null safety and operations: Declaring a variable with a ? after it means it is allowed to be null or might be null. Using ! after a variable name in use means to assert the variable is not null before use. Using ?? after a variable definition means to set the value to the value after the ?? if the value is null. For example, int x = y ?? 2; . If y were null, x would be equal to 2. If y were not null, x = y.

debugPrint: used for printing a message to the console.

Navigation with Named Routes: Opening and closing screens within the UI is accomplished with named routes and the Navigator function. The functions pushNamed and pop are used to open a named route and dispose of the current page. The new page is “pushed” on top of the current one, and “popped” off the top when disposed. Routes are stored in **lib/utility/src/RouteGenerator.dart** (*Navigation and Routing*, n.d.)

## Keychain

KeyChain is a flutter plug-in used to secure local storage of string data via Keychain and Keystore. Keychain is used for iOS (*Keychain Services*, n.d.), and Keystore for Android. Per the plug-in literature, Advanced Encryption Standard (AES) encryption is used for Android and a secret key is encrypted with (Rivest–Shamir–Adleman) RSA and the RSA key is stored in KeyStore.

The primary function of Keychain within the application is to store the username and password in a secure manner, during the current application session. After the user submits the username and password in the sign-in screen, the program attempts to verify these credentials can log into the associated email account. When it is proven it can, the program stores the username and password with the Keychain plug-in.

When the program needs to obtain the username and password later for other email operations, they can be obtained from the Keychain when requested.

## Implementing Google Cloud Vision Application Programming Interface (API)

Leveraging the work of the Summer 2022 cohort, and continuing their implementation of Google Cloud vision, we felt like the previous groundwork and effort would allow us to build upon their successes.

Their was one notable improvement that we identified. In the text recognition request, we opted to make a few changes. First, the previous group decided to use TEXT\_DETECTION which is according to Google's documentation, extracts text from any image, for example a street or traffic sign. However, after testing it was determined that it would be better to use DOCUMENT\_TEXT\_DETECTION because it is optimized for dense text and documents which fits the requirements of our project more closely.

var response = await api.annotate(BatchAnnotateImagesRequest.fromJson({  
 "requests": [  
 {  
 "image": {"content": image},  
 "features": [  
 {"type": "DOCUMENT\_TEXT\_DETECTION"},  
 ],  
 "imageContext": {  
 "languageHints": ["en", "es"]  
 }  
 }  
 ]  
}));

The second change highlighted above is the addition of languageHints. In some instances, the addition of languageHints has provided improved recognition when including this parameter. However, this is a new addition, and we are still testing to determine the level of increased effectiveness. The included hints are for English (en) and Spanish (es).

## Implementing Digest Email Parsing/Internet Mail Application Protocol (IMAP)

When a user initially logs in, or opens the application, and their sign-in credentials are validated, class main located in **lib/main.dart** will call function CacheService.updateMail. CacheService is a class located in **lib/services/cache\_service.dart,** and method .updateMail calls CacheService.getInstance().fetchAndProcessLatestMail. getInstance gets an instance of class MailFetcher, MailPieceStorage, and MailNotifier. fetchAndProcessMail reviews the MailPieceStorage last timestamp and then calls for the instance of MailFetcher to run method fetchMail() to obtain all USPS informed delivery emails since the last stored date.

MailFetcher is a class located in **lib/services/mail\_fetcher.dart.** .fetchMail() uses email utility MailUtility to obtain emails since the provide last timestamp. For each email it gets from the server for the associated user, it will process the email using the \_processEmail method where it will call \_getAttachments and get all image files that do not have an image content-id containing "ra\_", which USPS uses for "ride along" images that are not the scanned mail piece. For each scanned mail piece image that is found per email, a mailpiece object is created from the email and mailpiece image content using the \_processMailImage method. Method \_getOcrScan is used within \_processMailImage to call an instance of CloudVisionApi to obtain the image recognition results.

Images for each mail piece are not stored in the database. This is due to the potentially large storage size on the disk if the user were storing mail piece images from the past few years of USPS informed delivery emails. The scanned images and associated "Learn More" and "Set a Reminder" links are to be retrieved when the mail piece is retrieved at a later time using the mail piece search and view individual mail piece features.

## Implementing Mail View

### Mail Actions

On each individual mail piece view there will be a section after the image of the mail and sender/receiver info, which represents the Mail Actions panel. Within the Mail Actions panel there will be a list of links that are clickable and will open a browser within the MailSpeak app and launch the URL. These URLs will be parsed from either a QR code or barcode utilizing the Barcode Scanner API or adding the URLs parsed from the text utilizing Link Well. In the case there are zero links the section will not show.

The way this is implemented is during the same logic of **Section 6.8.** When the user logs into the system the CacheService is called where it calls MailFetcher and MailPieceStorage. MailFetcher will call the \_processMailImage method, which access OCR scan results and LinkWell plugin code to extract and compile a list of URLs. LinkWell utilizes regex to decipher between emails and phone numbers. This method will return a new MailPiece object to be saved into the database. It should be mentioned that the MailPiece model has an optional parameter for links. MailPieceStorage is where we store the MailPiece into the database with the \_saveMailPiece method.

For retrieving the data to be displayed, the user can search for mail and it will query the database based on the search parameters. This search parameters will return a list of mail pieces that include a list of the links that were generated by the cache service. These results are then passed along to the mail\_view\_indv UI page.

### Email addresses and phone numbers

On each individual mail piece view there will be a section after the image of the mail and sender/receiver info, which represents the Mail Actions panel. Within the Mail Actions panel there will be a list of emails and phone numbers that are clickable and will open your preferred mailing service or the phone app within the MailSpeak app. These emails and phone numbers will be parsed from either a QR code or barcode utilizing the Barcode Scanner API or adding the emails and phone numbers parsed from the text utilizing Link Well. In the case there are zero emails or phone numbers the section will not show.

The way this is implemented is during the same logic of **Section 6.8.** When the user logs into the system the CacheService is called where it calls MailFetcher and MailPieceStorage. MailFetcher will call the \_processMailImage method, which access OCR scan results and LinkWell plugin code to extract and compile a list of emails and phone numbers. LinkWell utilizes regex to decipher between emails and phone numbers. This method will return a new MailPiece object to be saved into the database. It should be mentioned that the MailPiece model has an optional parameter for emails and phone numbers. MailPieceStorage is where we store the MailPiece into the database with the \_saveMailPiece method.

For retrieving the data to be displayed, the user can search for mail and it will query the database based on the search parameters. This search parameters will return a list of mail pieces that include a list of the emails and phone numbers that were generated by the cache service. These results are then passed along to the mail\_view\_indv UI page.

### Sender Detection

Detecting and displaying the sender of each mail piece is crucial to making the UI intuitive for the user. Mail pieces are fetched from the provided email and processed using MailFetcher#processMailImage. We attempt to determine the sender in a number of ways. Our logic can be deduced through a series of questions: did Informed Delivery identify a sender? If not, can Google Vision’s Optical Character Recognition (OCR) identify the name of the sender? If a name cannot be deduced, can the logo be used for identification?

Since OCR is entirely dependent on the quality of the mail piece scan, we prefer to use the sender provided by Informed Delivery. We accomplish this by parsing the HTML of the digest email and searching for the <strong> tag. Since multiple mail pieces can be sent in one digest email, we deduce if the <strong> tag correlates to the image currently being processed via their relative positions in the HTML. If a sender was not provided by USPS we attempt to extract the information from the scanned image via OCR processing.

As mentioned previously, OCR processing is not always reliable and is heavily dependent on the quality of the scan. Due to this, we adopted a primary and secondary method for sender identification via OCR. The primary method is to take the name from an address read from the image. The issue we ran into here was that return addresses are not always written in the same location. In classic mail, the return address is always at the top left of the envelope, however, this is not the case when businesses send advertisements. Luckily, they’re generally placed above the sender address on the mail piece. So, we simply take the first address read by the scan and check to see if a name value was parsed from the image.

If the scan was unable to read the address or deduce the name from it, we move onto logo recognition. Google’s Vision API is extremely robust and instantly matches a name to identified logos when scanning an image. This proved fairly trivial to implement. Unfortunately ,we found that poor scans affect the logo far more severely than the text. The best case scenario is the logo is successfully interpreted. For our purposes, the worst case scenario is when a logo is *mis*interpreted, rather than simply not being interpreted. In our testing we found that the graininess of one scan caused the Vision API to think a Bank of America logo was the Merrill Lynch logo. This can potentially create a negative experience for the user, however, we are limited by the technology in this case.

### Displaying Individual Mail Pieces

After a user reviews their mail piece search results and clicks on the mail piece in the list view screen, they are brought to the mail piece individual screen. The code for the UI is contained within file **lib/ui/mail\_view\_indv.dart**, which defines class MailPieceViewWidget.

The screen consists of displaying data stored with the mail piece, such as the sender (if known), the mail piece time stamp, and Google Vision scanned image OCR data. To save memory stored on the user's local device, only the mail piece scanned image content-id is stored with a mail piece, not the image itself. Using this content-id, and date of the mail piece, the original USPS informed delivery email is retrieved from the user's email account. The OCR data is also stored with the mail piece to limit the amount of mail piece scans are implemented by the program, as these are extra cost.

Initially the selected mailPiece object from the list view is passed to the MailPieceViewWidget. This widget has been defined as a stateful widget to allow it to update data and work more seamlessly with the preceding screen. Initially, when the screen is first loaded, it calls function \_getMailPieceEmail which attempts to get the original Informed Delivery email of the mail piece using methods from class MailFetcher. After obtaining the email, the method \_getImgFromEmail is executed to get the scanned image. If a mailpiece does not have any ids, it means it was likely uploaded manually and the email retrieval function is bypassed.

Every mail piece with an id has exactly one associated USPS informed delivery email date. Each mail piece scanned image also has a unique content-id from USPS. The email is obtained as type MimeMessage, which has parts including the "text/Hyper Text Markup Language (html)" part, and embedded "images" as separate parts. Multipurpose Internet Mail Extensions (MIME) defines the protocol standards that the data is encoded in. Due to email mostly being text, images are stored as Base64 which is essentially an ASCII text representation of the data, which must be converted back to binary data to be displayed as an image in the application using the Image.memory(base64Decode(picture) code. Due to the delay in obtaining the email, \_getMailPieceEmail and \_getImgFromEmail are asynchronous functions, and the mail piece image a late Image type which is generally loaded a couple seconds after the screen is displayed. A loading screen is displayed while this operating is processing.

After retrieving the image, method \_getLinkHtmlFromEmail is called. Using the content-id, the associated position of the mail piece "Set a Reminder" and if available, "Learn More" links can be established within the "text/html" part of the MimeMessage. There is distinctive text that can be obtained from the "Set a Reminder" and "Learn More" links in the html that can be used for retrieval, including finding the associated embedded URL link. One caveat to finding this is that the USPS email type is "quoted-printable", and the html part must be decoded as such to have an html query selector search all the available attributes in the html of each embedded link. This is done using the decodeText ( ContentTypeHeader ('text/html'), 'quoted-printable') ) code.

Method \_getLinks is utilized to obtain a functional URL from the embedded links. The links are displayed in the screen like way they are displayed in the Informed Delivery emails.

When the operations to get the mail image and links are finished, attribute "loading" is set to false which removes the loading mail piece screen and displays the mail piece display screen.

### Email Services

All email retrieval is consolidated to a single class, for ease of management. The code for this service is contained within file **lib/services/mail\_utility.dart**, which defines class MailUtility. This class primarily does three things. It fetches a list of emails since a certain date, with subject filter, and sender filter, using method \_getEmailsSince. This is primarily how the initial mail pieces are obtained.

It also has a method\_getEmailOn where it fetches a specific email based on a target date, subject filter, and sender filter. This is primarily how additional information is retrieved for each mail piece when it is viewed on the application.

It also checks that the provided username and password during sign-in actually can log into the email account using method \_getImapClient which returns a bool, either true or false.

Methods \_getEmailsSince and \_getEmailOn obtain the username and password from the Keychain utility at time of request for emails. \_getImapClient is used to validate the username and password, as such this method does not utilize the Keychain.

## Implementing Notifications

**Approach and Reason**

The notification screen was implemented to allow the user to be able to keep track of what's new and to keep the user updated with the latest mail piece of their choice. The first approach that we had was to ensure the user can navigate to the notifications view easily by allowing the user to select the notifications card from the mail menu. The other approach was to create a manage notifications tab to allow the user to get notifications by the keywords that they entered. Instead of getting every mail piece, users are able to get the mail piece of their choice by the keywords that they entered. We also implemented a ‘Go To Message’ button in the notifications view so that the user can look at the specific mail piece by just clicking on the button without having to navigate away from the notifications view. We also implemented the ‘Clear’ and ‘Clear All’ features in the notifications view to allow the user to delete or keep the notification of their choice. The ‘Manage Notifications’ view was also designed with a ‘Delete’ and ‘Add’ button to allow the users to enter or delete the keyword of their choice.

### Notifications View

The notifications screen displays a list of notifications. We wanted the main view to be divided into two tabs, Notifications and Manage. The Notifications tab will show notifications by keyword text, go to message button and clear button. The Manage tab will be where the user can add or delete a notification subscription. The code for this view is \lib\ui\notifications.dart.

Implementing these views were a bit tricky because of having a TabBarView, for some reason displaying lists were not correctly rendering. The work around for this issue was to create a for loop to iterate the Notification lists. For each iteration, we would display the Date as a text field, Keyword as a text field, Go To Message button and clear button. The Go To Message button navigates to the mail that contains the Notification Subscription Keyword.

### Create Notifications Subscription

In the Notifications Manage Tab, there is a Keyword Text field you type into and click on Add to create a notification subscription. When you click on Add, there's a createSubscription method that gets called in \lib\services\mail\_notifier.dart. This method inserts the Keyword into the NOTIFICATION\_SUBSCRIPTION database table. The code for this function is \lib\ui\notifications.dart.

### Delete Notifications Subscription

In the Notifications Manage Tab, there is a list of notification subscriptions being displayed. For each row, it shows Keyword and a red Clear button. When you click on Clear, there's a removeSubscription method that gets called in \lib\services\mail\_notifier.dart. This method deletes the Keyword from the NOTIFICATION\_SUBSCRIPTION database table filtered on Keyword. The code for this function is \lib\ui\notifications.dart.

### Clear Notification

In the Notifications Tab, there is a list of notifications being displayed. For each row, it shows Keyword, Go To Message button, and a red Clear button. When you click on Clear, there's a clearNotification method that gets called in \lib\services\mail\_notifier.dart. This method deletes the Keyword from the NOTIFICATION\_TABLE database table filtered on MailPieceID and Keyword. The code for this function is \lib\ui\notifications.dart.

### Notification Management View

The notification management screen displays a list of notification subscriptions. This view can be seen by clicking on the Manage Tab within the main Notifcations View. The Manage tab will show notification subscriptions by keyword text and a clear button. The Manage tab will be where the user can add or delete a notification subscription. The code for this view is \lib\ui\notifications.dart.

### Notification fetch process

Fetching the notifications process was implemented in \lib\services\mail\_notifier.dart.There are two methods, getSubscriptions and getNotifications. Both functions have a simple select query that retrieves data from NOTIFICATION\_SUBSCRIPTION AND NOTIFICATION\_TABLE and returns a List of Keywords. This Notification list is being displayed in the main Notification View and the Notification Subscription list is being displayed in the Notification Management View.

### Navigate to mail item from notification

Navigation from a mail item is very simple. When the user selects a notification, the application will get the resulting mailpiece via the mailpiece id. This is then passed into the Mail Piece View which will then display the notified mail piece. It was decided to reuse the Mail Piece View as it is a common page for displaying mail pieces which results in much less duplicated code as well as less developer confusion. For example, if you needed to change what the mail piece displayed, it would only need to be changed on one page instead of four.

## Digest Modification

### Digest Modifications

The Digest page was existing functionality however it was decided that it should be reskinned to match the rest of the application. Many of the elements such as the top and bottom bar were changed to match the rest of the application. There were two major changes to the digest. The first and most major was that the digest page will now pick the latest digest instead of the digest for the day you are currently on. This involved some changes to how the application pulls the digest as we now needed to pull all digests then order them by date instead of only getting digests for the current day. It was originally thought that we could use the SORT IMAP function, however that is not well supported so it was decided to do our own sorting. This is a possible future enhancement if the SORT function becomes better supported in the future.

The other major change was to convert the All Details button to navigate to the Mail Piece page for the selected mail piece. This replaced old functionality that would read out data on the selected mail piece, and this new functionality provides more information to the user. We use the Mail Piece page to avoid having duplicate information on the Digest page which already exists on the Mail Piece page.

## Implementation of Voice Assistants

### Android Assistant Integration

Google Assistant integration was implemented by responding to intents that the Google Assistant sends to the application. Google Assistant knows what capabilities the application has by the list of capabilities in the shortcuts.xml file. In this case the application has the capability to "Get\_Thing", "Create\_Thing" and “Open\_App\_Feature”. Get thing corresponds to the Search and Create thing corresponds to adding notifications. Open App Feature allows the user to open the application directly to the Notifications page. These are defined in the native code. We use only Built-In-Intents to avoid having to provide training phrases as you would need to do for custom-intents.

In the shared code, a service called assistantService was created. This handles the processing of intent data into ApplicationFunctions which will tell the application what to do. For example, if the GET\_THING intent is received then we check if the query is for digest or another word. If it is digest, we create a ApplicationFunction to navigate to the digest page, if it is another word, we create an ApplicationFunction to search for the given keyword. It was decided to make this its own service, since intents can be received at any point in the application, and we would not want to duplicate the code on every page. It was decided to use ApplicationFunctions, since that is what the chatbot uses so less code would need to be created specifically for Google Assistant.

Another bit of shared code is the assistant\_state. This serves as a base class for all UI pages in the application. Using this, the application subscribes to intents and acts upon them when received. This base class was created since all of the pages must have this basic logic, and it is easier to have it in one place and override where necessary instead of duplicating the code on each page. A good example of overriding is on the search page, we override the processFunction method so we add the search terms on that page, instead of navigating to the search page again even though we are already on it like the base code would do.

The application uses the package "Recieve\_Intent" to act on the intents that are sent to the Android application. On every page we subscribe to the intent stream using this package in the assistant\_state.dart file. When the application starts up, this package is also used to find the starting intent if one exists in main.dart. If it does, we will then create an applicationFunction using the assistantService and pass that in as an argument to one of the two starting pages (MainWidget or SignInWidget). Those pages will then handle the argument like they would if they received the intent in the intent stream.

One major problem we ran into is that Google Assistant requires the app to be uploaded to the play store to work, and their testing tools other then ADB commands do not work for Flutter projects. When uploading to the play store, Google Assistant also requires app actions to be reviewed before the general public and use them. Testers can access them early via the internal test track by joining a Google Group, but it still did not work and Google Assistant acted as though the application did not exist. At this point in time, we are waiting for our Google Review. Another issue we ran into was using the proper built-in intents. We were unable to implement removing a notification keyword because Google Assistant does not provide an intent for removing something, only adding. There were also issues with date ranges for searching as Google Assistant provides everything via keyword so there was no way to parse the date ranges in the time provided to the project since the user could say anything.

# Project File Structure

The use of a Model-View-ViewModel (MVVM) is an established architectural pattern in software development. One of the key distinctions, and benefits with this framework is that it allows architects to segregate business processes or logic from user interface (UI). This allows for changes to the UI without having to also modify the underlying data model and the same in reverse. This process can dramatically speed up development when everything is working correctly.

According to Jitesh Mohite, there are three key aspects when MVVM is working correctly (Mohite, 2020).

1. **Maintainability**: code is easier to maintain and reuse since the presentation and logic are loosely coupled.
2. **Testability**: a ViewModel is easier to test than other types of code such as event-driven.
3. **Extensibility**: this type of architecture provides the ability for the system to evolve over time as the needs change.

## Overview

The three components of this architecture are the Model, ViewModel, and the View.

|  |  |  |
| --- | --- | --- |
| Model | ViewModel | View |
| The Model represents the business logic, code validation and everything else about that specific item. Models can refer to services, repositories, APIs, etc. | The ViewModel is the go between passing information back and forth from the Model and View to each other. A ViewModel can interact with one or more views. | The view is how the user interacts with the widgets shown on the screen. The user through an event requests some action which interacts with a ViewModel to complete the task. Eventually, the ViewModel updates the view. |

Table - Model-View-ViewModel

Within every Flutter application there is a lib folder that contains the underlying code for the entire application. Within this app, our folder structure is composed as follows:

* lib – root
  + email\_processing
  + exceptions
  + generated
  + image\_processing
  + models
  + services
    - bases
  + ui
  + utility

As mentioned above, our **Model** classes are broken down into the following items each controller's access to the specific item referenced by the filename:

* Address
* Address.g
* AppplicationFunction
* Auguments
* Code
* Code.g
* Digest
* EmailAuguments
* Logo
* Logo.g
* MailPiece
* MailResponse
* MailResponse.g
* MailSearchParameters
* Notification
* NotificationSubscription
* SearchCriteria

## Pubspec.yaml

A requirement of all Flutter projects, pubspec.yaml file is located at the top of the project tree and contains metadata about the project that the Dart and Flutter tooling needs to know. This file contains a mix of autogenerated and manually configured content, all of which is important to the entire project.

The following information is specific to this project. For general information about pubspec.yaml, please refer to the following the official documentation on Flutter's site. <https://docs.flutter.dev/development/tools/pubspec>

### Description

The top of the file contains information about the project.

### Dependencies

The following dependencies have been installed based on packages that were required by the application. There dependencies represent plugins which are a special kind of package that makes platform functionality available to the app. It is a piece of software that adds capabilities to your app. Plugin packages can be written for Android, iOS, web, macOS, Windows, Linux, or any combination thereof. Each of these are added to the project automatically by navigating to the terminal and typing at the command prompt > flutter pub add {name of the package}.

|  |  |
| --- | --- |
| Dependency | Description |
| **camera: ^0.9.8+1** | A Flutter plugin for controlling the camera. Supports previewing the camera feed, capturing images and video, and streaming image buffers to Dart |
| **cupertino\_icons: ^1.0.2** | Default icons asset for Cupertino widgets based on Apple styled icons |
| **enough\_mail: ^2.1.1** | IMAP, POP3 and Simple Mail Transfer Protocol (SMTP) for email developers. Choose between a low level and a high level API for mailing. Parse and generate MIME messages. |
| **firebase\_analytics: ^9.3.7** | Flutter plugin for Google Analytics for Firebase, an app measurement solution that provides insight on app usage and user engagement on Android and iOS. |
| **firebase\_core: ^1.24.0** | Flutter plugin for Firebase Core, enabling connecting to multiple Firebase apps. |
| **flutter\_chat\_ui: ^1.6.4** | Actively maintained, community-driven chat User Interface (UI) implementation with an optional Firebase BaaS |
| **flutter\_datetime\_picker: ^1.5.1** | A date time picker for Flutter, you can choose date / time / date&time in English Dutch and Chinese, and you can also custom your own picker content |
| **flutter\_secure\_storage: ^5.0.2** | Flutter Secured Storage provides API to store data in secure storage. Keychain is used in iOS, KeyStore based solution is used in Android. |
| **flutter\_svg: ^1.1.0** | An SVG rendering and widget library for Flutter, which allows painting and displaying Scalable Vector Graphics 1.1 files. |
| **flutter\_tts: ^3.5.0** | A flutter plugin for Text to Speech. This plugin is supported on iOS, Android, Web, & macOS. |
| **flutter\_typeahead: ^4.1.1** | A highly customizable typeahead (autocomplete) text input field for Flutter |
| **get\_it: ^7.2.0** | Simple direct Service Locator that allows to decouple the interface from a concrete implementation and to access the concrete implementation from everywhere in your App" |
| **global\_configuration: ^2.0.0-nullsafety.0** | A flutter package for managing different configurations by merging them together and making them available everythere inside the app via a singleton |
| **google\_mlkit\_barcode\_scanning: ^0.3.0** | A Flutter plugin to use Google's ML Kit Barcode Scanning to read data encoded using most standard barcode formats. |
| **googleapis: ^9.1.0** | Auto-generated client libraries for accessing Google APIs described through the API discovery service. |
| **googleapis\_auth: ^1.3.1** | Obtain Access credentials for Google services using OAuth 2.0 |
| **http: ^0.13.4** | A composable, multiplatform, Future-based API for Hypertext Transfer Protocol (HTTP) requests. |
| **image\_gallery\_saver: ^1.7.1** | A new flutter plugin project for save image to gallery, iOS need to add the following keys to your Info.plist file. |
| **image\_picker: ^0.8.5+3** | Flutter plugin for selecting images from the Android and iOS image library, and taking new pictures with the camera. |
| **intl: ^0.17.0** | Contains code to deal with internationalized/localized messages, date and number formatting and parsing, bi-directional text, and other internationalization issues |
| **json\_annotation: ^4.5.0** | Defines the annotations used by json\_serializable to create code for JSON serialization and deserialization. |
| **json\_serializable: ^6.2.0** | Automatically generate code for converting to and from JSON by annotating Dart classes. |
| **layout: ^1.0.3** | Layout encourage consistency across platforms, environments, and screen sizes by using uniform elements and spacing. |
| **loader\_overlay: ^2.0.7** | A simple package to simplify screen management. When loading any async task, this package prevent the user from interacting with the screen until the async task finishes |
| **path\_provider: ^2.0.11** | Flutter plugin for getting commonly used locations on host platform file systems, such as the temp and app data directories. |
| **path\_provider\_android: ^2.0.11** | Flutter plugin for getting commonly used locations on host platform file systems, such as the temp and app data directories. |
| **permission\_handler: ^10.0.0** | Permission plugin for Flutter. This plugin provides a cross-platform (iOS, Android) API to request and check permissions. |
| **receive\_intent ^0.2.2** | A Flutter plugin to receive intents in the Android application, and pass them to the Dart application. Used for Google Assistant. |
| **speech\_to\_text: ^5.6.0** | A Flutter plugin that exposes device specific speech to text recognition capability. |
| **sqflite: ^2.1.0** | Flutter plugin for SQLite, a self-contained, high-reliability, embedded, SQL database engine. |
| **sqlite3\_flutter\_libs: ^0.5.10** | Flutter plugin to include native sqlite3 libraries with your app |
| **syncfusion\_flutter\_datepicker: ^20.2.50** | The Flutter Date Range Picker widget allows users to easily select dates or a range of dates. It has four built-in views that allow quick navigation to the desired date. |
| **toggle\_switch: ^2.0.1** | Toggle Switch - A simple toggle switch widget. It can be fully customized with desired icons, width, colors, text, corner radius etc. It also maintains selection state. |
| **url\_launcher: ^6.1.4** | Flutter plugin for launching a URL. Supports web, phone, Short Message Service (SMS), and email schemes |
| **uuid: ^3.0.6** | RFC4122 (v1, v4, v5) UUID Generator and Parser for all Dart platforms (Web, VM, Flutter) |
| **xml: ^6.1.0** | A lightweight library for parsing, traversing, querying, transforming and building XML documents. |

Table - Dependancies

### Assets

The assets section is where static files that are used by the application are defined. This includes the large icons for the main menu screen and the smaller icons for the top appBar and the BottomNavigationBar.

|  |  |  |
| --- | --- | --- |
| Image | Filename | Usage |
| N/A | - assets/credentials.json |  |
|  | - assets/icon/icon.png | This is the image used on the device as the icon. |
|  | - assets/icon/settings-icon.png | This is the small icon used on the appBar. |
|  | - assets/icon/calendar-icon.png | This is the small icon used on BottomNavigationBar. |
|  | - assets/icon/exit-icon.png | This is the small icon used on the appBar. |
|  | - assets/icon/home-icon.png | This is the small icon used on BottomNavigationBar. |
|  | - assets/icon/notifications-icon.png | This is the small icon used on BottomNavigationBar. |
|  | - assets/icon/scanmail-icon.png | This is the small icon used on BottomNavigationBar. |
|  | - assets/icon/search-icon.png | This is the small icon used on BottomNavigationBar. |
|  | - assets/icon/back-icon.png | This is the small icon used on the appBar. |
|  | - assets/icon/chatbot-icon.png | This is the small icon used on BottomNavigationBar. |
|  | - assets/icon/uploadmail-icon.png | This is the small icon used on BottomNavigationBar. |
|  | - assets/icon/search\_mail\_icon\_lg.png | This is the version of the icon used on the Main Menu. |
|  | - assets/icon/daily\_digest\_icon\_lg.png | This is the version of the icon used on the Main Menu. |
|  | - assets/icon/settings\_icon\_lg.png | This is the version of the icon used on the Main Menu. |
|  | - assets/icon/notification\_icon\_lg.png | This is the version of the icon used on the Main Menu. |
|  | - assets/icon/upload\_image\_lg.png | This is the version of the icon used on the Main Menu. |
|  | - assets/icon/scan\_mail\_icon\_lg.png | This is the version of the icon used on the Main Menu. |
|  | - assets/icon/applogo-mailspeak.png | This is the large logo used on email login screen. |
|  | - assets/icon/applogo-mailspeak-200.png | Smaller version of the logo used on the email logon screen when necessary. |

Table - Assets

# User Interfaces (UI)

## Front-End Screens

The user interface of MailSpeak utilizes Flutter's modular widgets to speed up design and development in order to compose a series of screens that help users quickly navigate around the application in order to accomplish the main purpose of the application: to help them navigate and get the most value out of the Informed Delivery email service provided by the United Stated Postal Service (USPS). There are two navigational elements that are consistent throughout the application, they are the referred to in Flutter as the appBar which is located at the top of the screen and the BottomNavigationBar located at the bottom of the screen. The appBar will contain a back icon capable of navigating the user to the previous screen and a logout icon to allow the user to exit the application. The BottomNavigationBar will contain a set of icons which duplicate the functionality contained on the main menu page as shown below

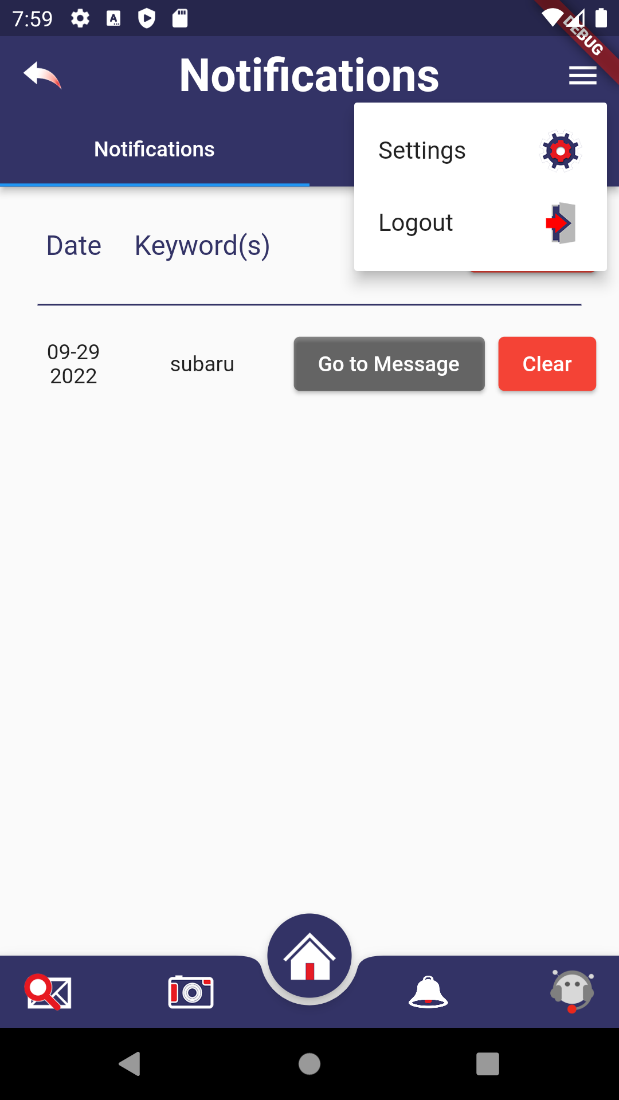


Figure - AppBar with Menu

Icon

Description automatically generated

Figure - BottomNavigationBar

The **Email Login Screen** provides a place for users to enter the credentials for their Informed Delivery registered email address. This will allow the application to poll the email services for Informed Delivery email messages in order for the application to pull those messages into the system. This user interface includes a checkbox which users must use to indicate that they have read through the terms and conditions and privacy policy of the application both of which are provided for the user.

A picture containing text, electronics, screenshot

Description automatically generatedGraphical user interface, text

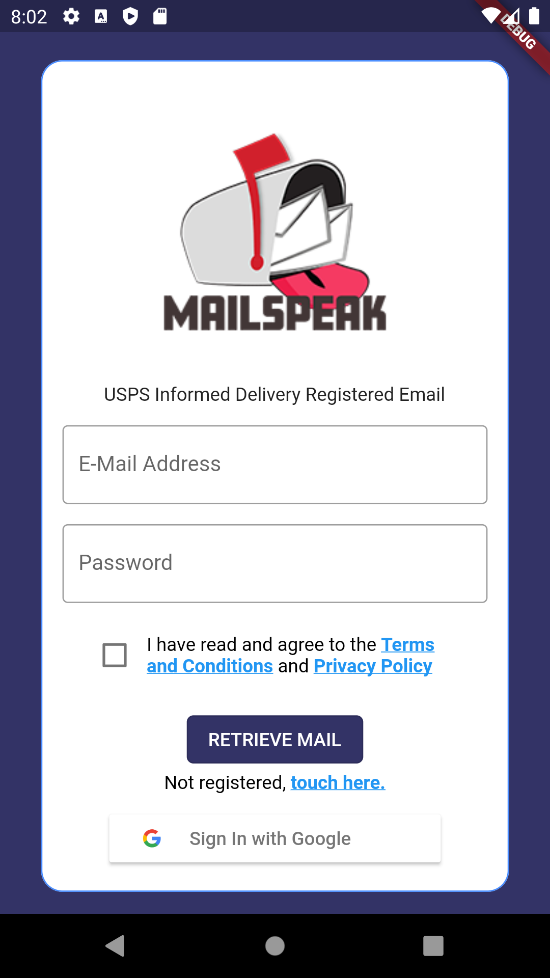
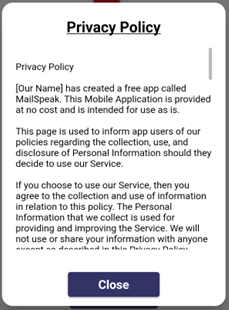
Description automatically generated

Figure - Email Credential Login Screen

The Main Menu provide access to all the functionality of the application. The main features of this screen are provided by the four buttons beginning with Search Main, Daily Digest Upload Main, Scan Mail.

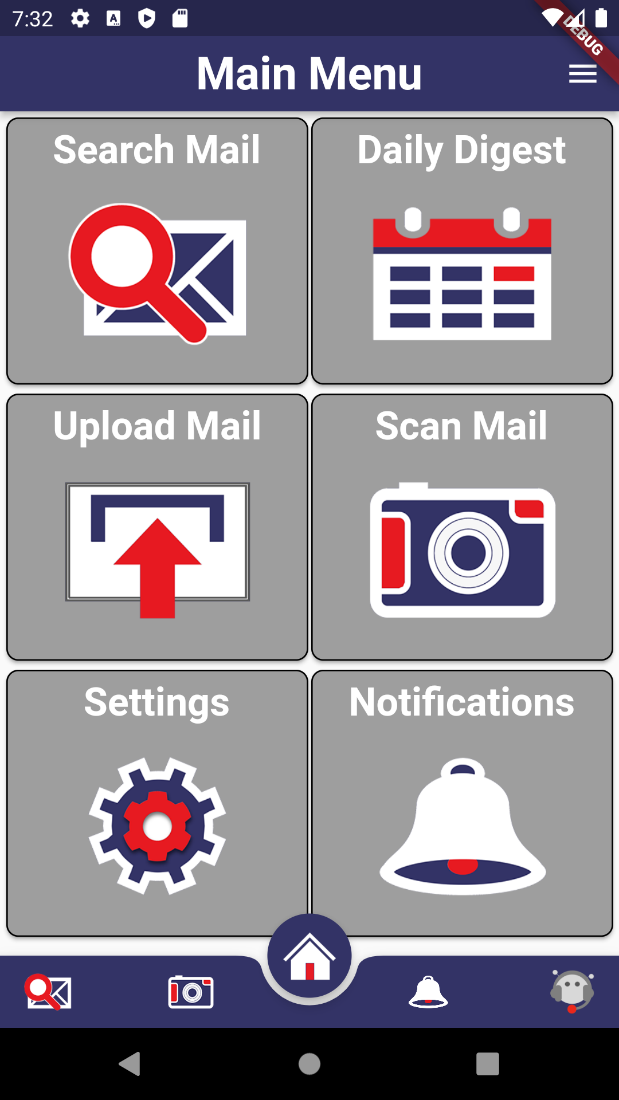
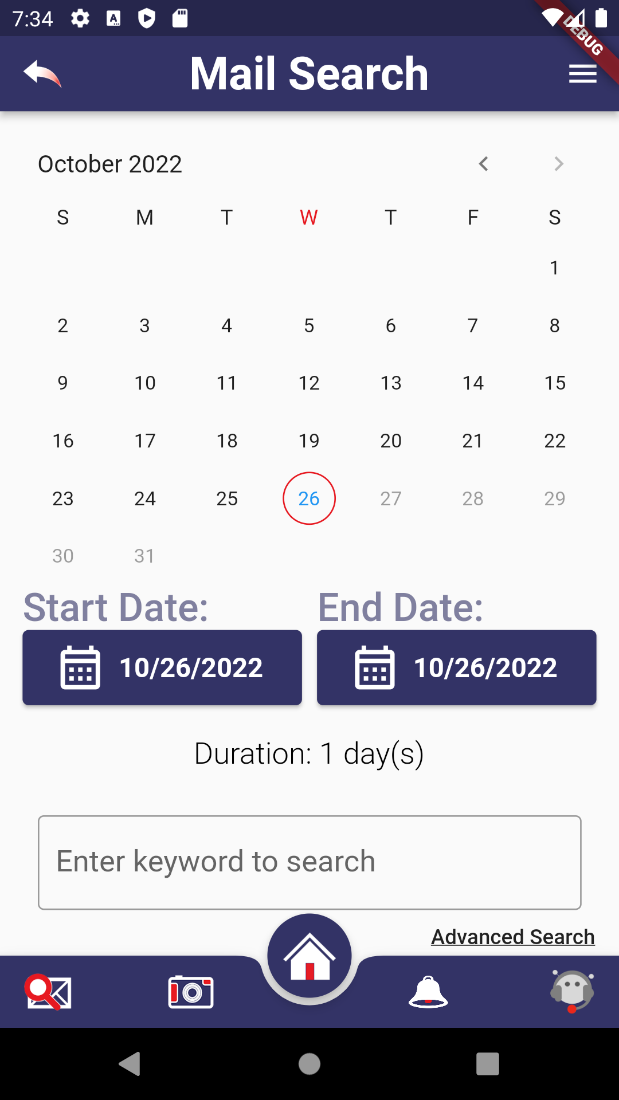


Figure - Main Menu Screen

The Mail Search provides users with the ability to search through Informed Delivery scanned mail pieces by date ranges and keywords which are used against a variety of text fields captured from a scanned mail piece.

Graphical user interface, application

Description automatically generated

Figure - Mail Search Interface

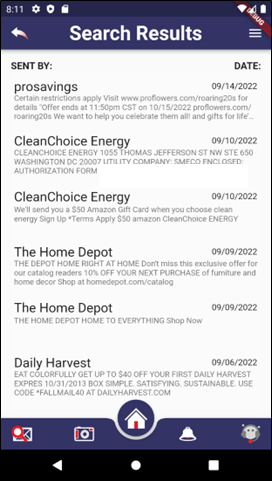


Figure - Mail Search Results

The Individual Mail View provides users with the ability view all content related to an individual scan of a physical piece of mail starting with the scanned image then moving onto the do more with your mail section, finally ending with the recognized text generated from sending the scanned image to Google Vision and parsing the returned results.

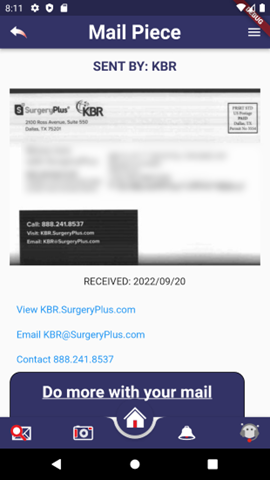
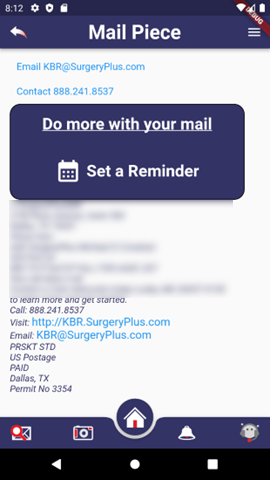
 

Figure - Mail Piece Screens

The Daily Digest allows a user to scroll through list of the scanned mail of the day. The links loads a dialog with any links that were present in for that item and the all details load the individual mail view for that is used by the search view.

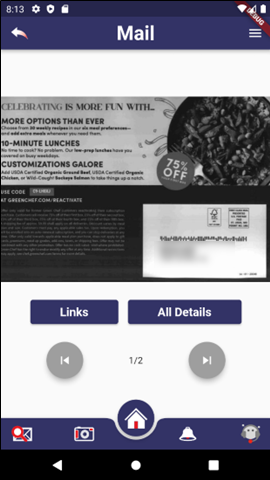
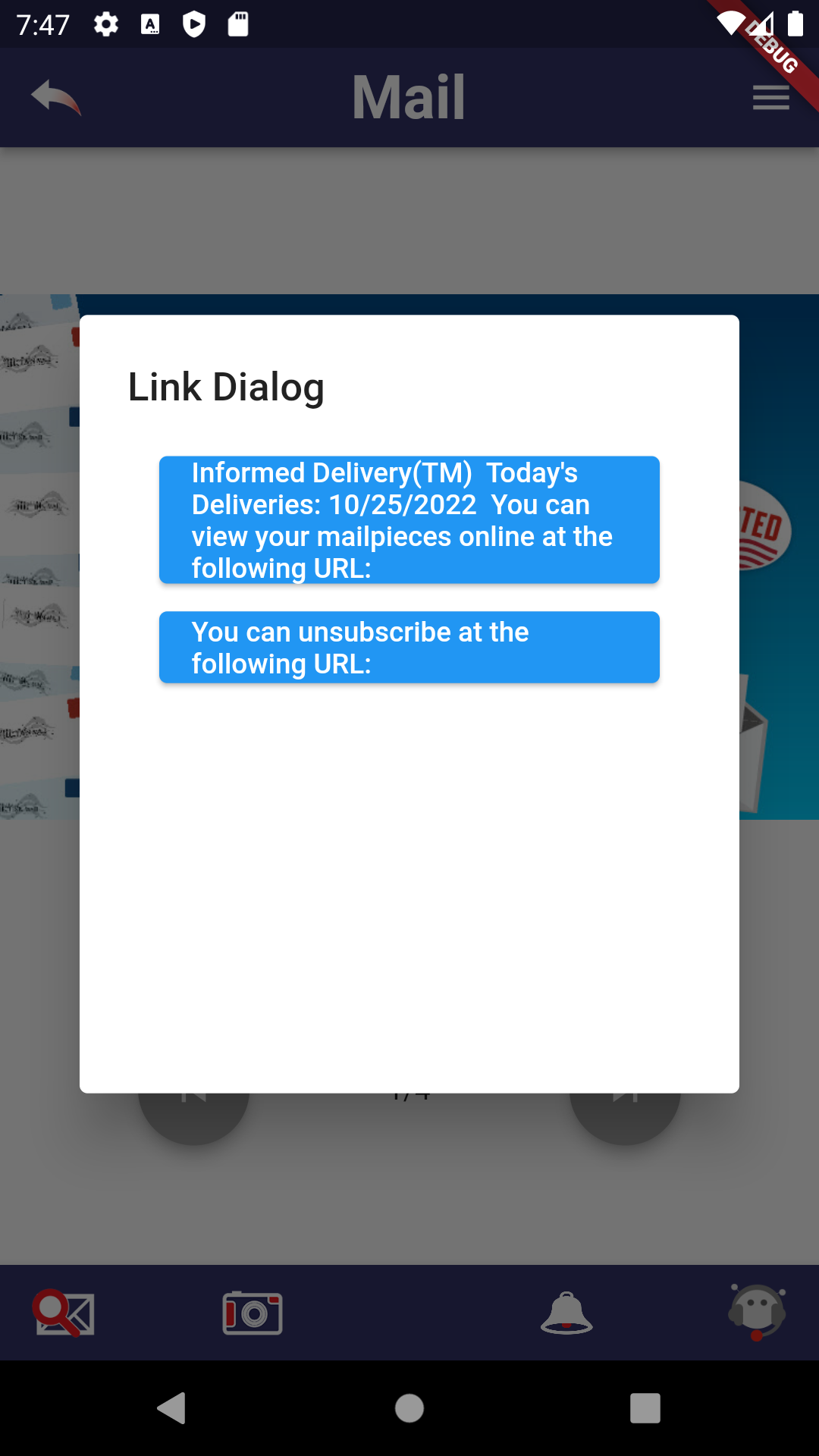
 

Figure - Daily Digest

Graphical user interface

Description automatically generated with medium confidence

Figure - Daily Digest Cont.

The Upload Mail provides a user with the ability to upload a picture of a piece of mail into the application. When the user selects this feature, the application opens up the devices photos folder to allow a user to select an image.

Graphical user interface, application

Description automatically generated

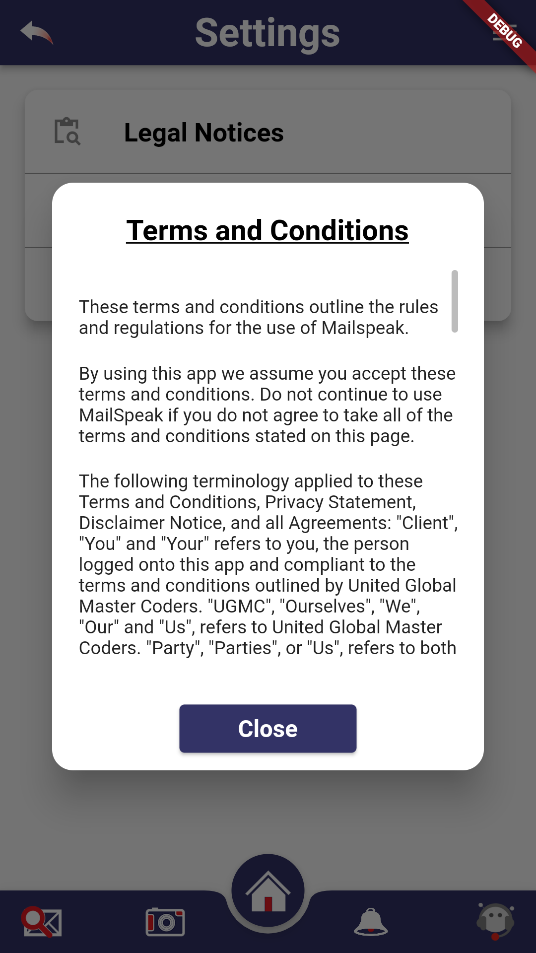
Figure - Upload Mail From Device Images

The Scan Mail feature allows a user to take a photo of a mail piece and immediately upload it into the application. When a user selects this feature, the application opens up the camera in preparation to take a photo and upload the image into the application.

A picture containing text, indoor, room, ceiling

Description automatically generated

Figure - Capture Mail with Camera

In the Settings Interface, the user can select which areas of a mail piece that an envelope should be evaluated by Google Vision. Additionally, the user can select email fields that are important for the screen reader.

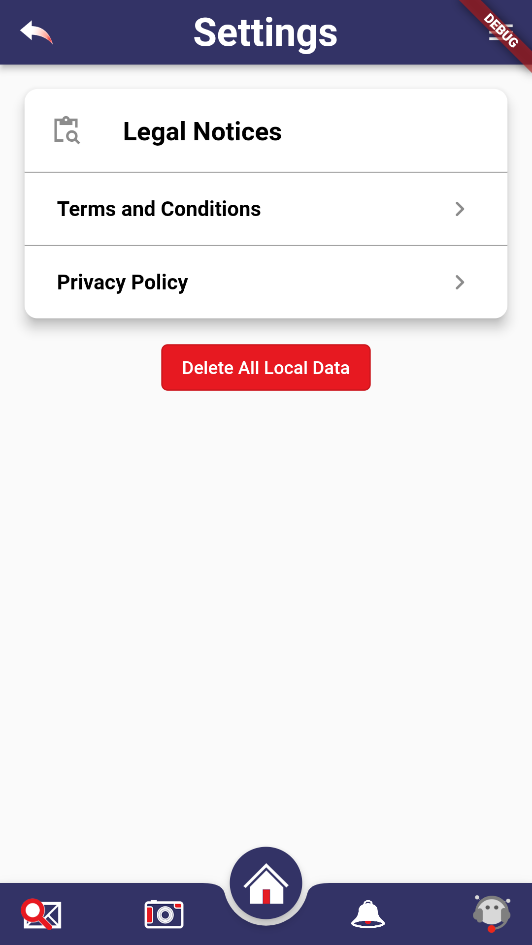
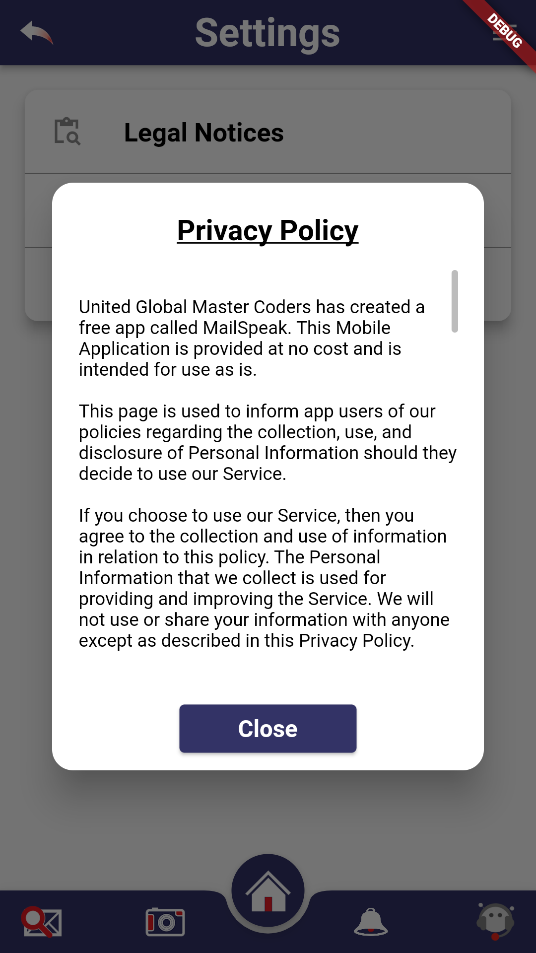


Figure - Application Settings

In the Notifications Feature, a user can see and manage their notifications. In the Notifications Display, this is where a user will be directed when a notification is triggered. From this area, they will have the option to navigate directly to the mail piece that triggered the alert or dismiss the notification. In the Manage Notifications, this is the area where users will be able to manage keywords to trigger notifications within the application.

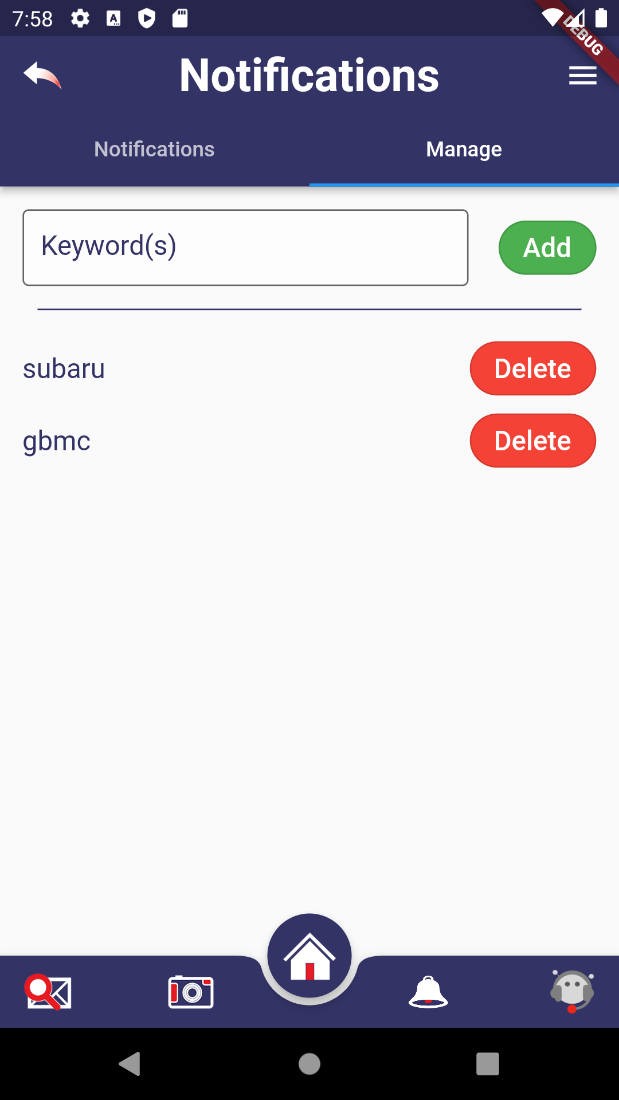
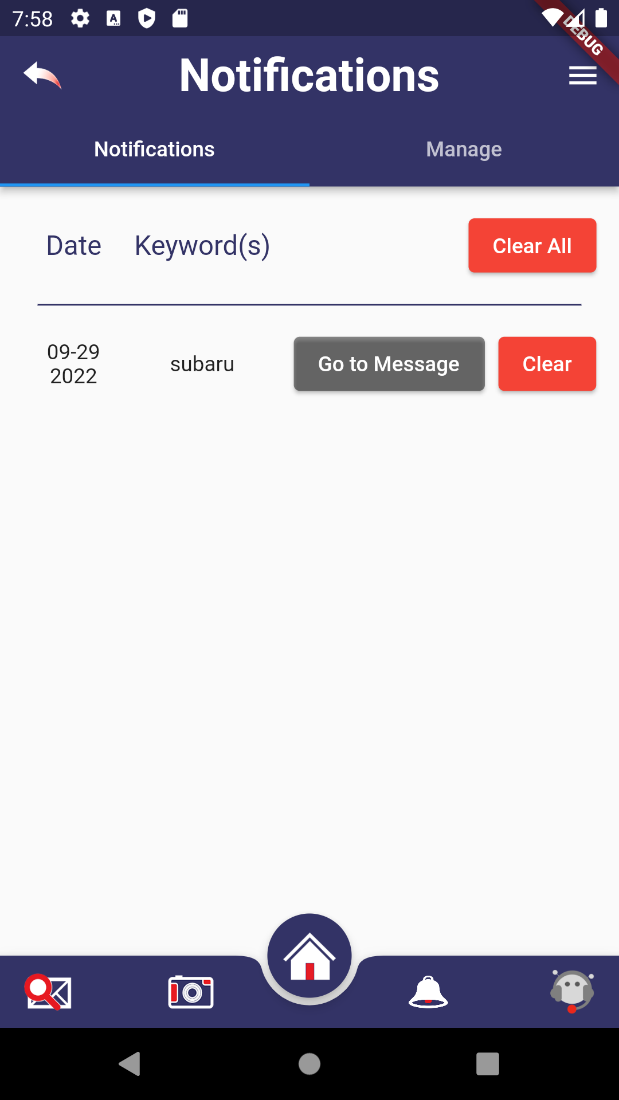


Figure - Notification Screens

In the Chat Support interface, users will be able to interact with an automated agent to provide an automated agent capable of launching application tasks such as search mail, set notification, and other system functionality.

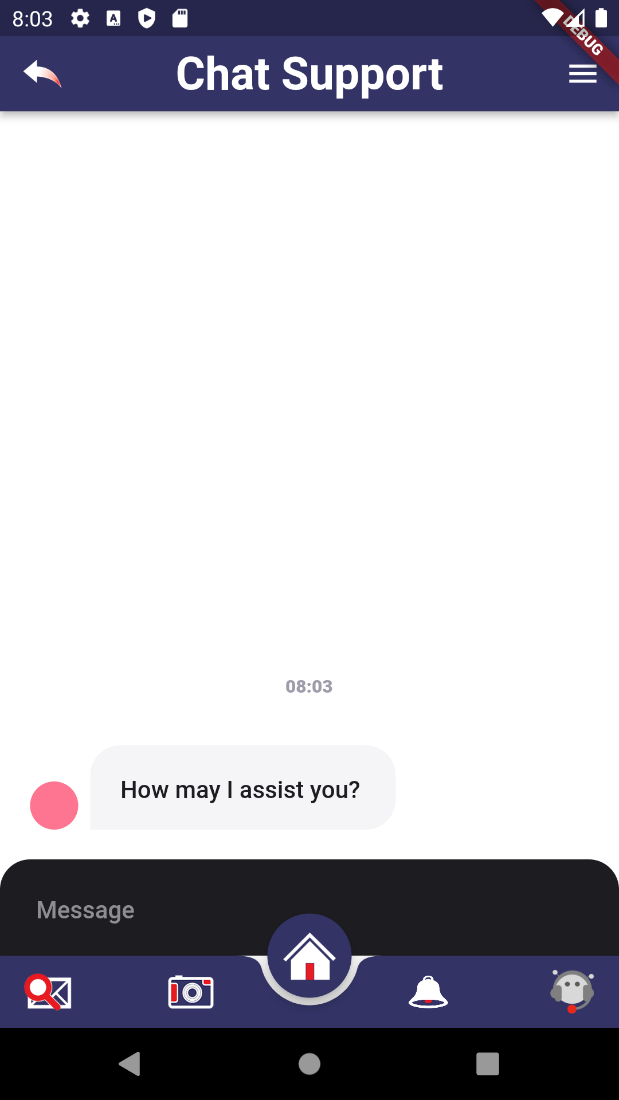


Figure - Chatbot Support Screen

# Accessibility

The following provides a roadmap for how we prioritized our approach to accessibility within MailSpeak for our users. The following table highlights the requirements for accessible software design based on Section 508 and Americans with Disabilities (ADA) Act standards where possible, we added our approach under the category column. We aimed to use the simplest language for instructions, prompts, and outputs, where possible.

|  |  |  |
| --- | --- | --- |
| Category | System | |
| Navigation | | The software provides keyboard equivalents for all mouse actions, including buttons, scroll windows, text entry fields, short-cut keys, and pop-up windows. |
| Semantics | | The software provides clear and precise instructions for use of all chatbot, gesture, voice-driven, and reading mode functions as part of the user documentation. |
|  | | The most common universal keyboard features were applied. In addition, users can use any third-party keyboard to maximize usability, even speech-to-text options. |
|  | | The software has a logical tabbing order among fields, text boxes, and focal points. Which minimizes the amount of information on each page (compared with a desktop or laptop). |
| Gestures | | Interactive elements are positioned where they can be easily reached regardless of how the device is held. Taking into consideration left- versus right-handed use, assumptions about thumb range of motion. Buttons are clearly labeled and explain their purpose. |
|  | | Simple gesture controls such as simple tap, left/right swipe gestures, pull down, swipe up, zoom in/out, etc. are in place of more complex ones.  Audio or video longer than three seconds can be paused or stopped by the user. |
|  | | Users can easily go back if unintentional actions such as accidental clicking, swiping, and tapping were performed. |
| Integration | | The software support existing accessibility features user may currently have built into their operating system: an on-screen keyboard, a Bluetooth keyboard, and text to speech, autofill, screen adjustments, etc.  We've reduced the amount of text entry required by providing select menus, radio buttons, check boxes, or by auto-filling known information (e.g., date, time, location). |
| Third-party Accessibility Software | | All visible text can be read by a screen reader i.e VoiceOver, TalkBack.  All interactions can be completed using only a screen reader.  The screen reader makes an announcement if the page content changes significantly (for instance, new search results have loaded, or page content has been filtered by some criteria).  All error messages are announced by the screen reader.  When content is deleted or removed from the page, the focus is moved backward to something previously seen rather than forwards.  Screen readers can focus and read disabled elements (for example, a disabled button).  Content that is inherently inaccessible, like drag and drop functionality that requires a mouse, has an alternative implementation that screen reader users can use.  Links have contextual labels that provide more information for screen reader users.  Content that expands and collapses allows the screen reader to read the current state for whether the content is expanded or collapsed.  Skip the navigation option is available, to avoid unnecessary information. |
| UI | | Allow sufficient response time to accommodate the slowest users. In addition, timed responses can modify by the user to adjust the timing parameters of any required timed responses.  Users have enough time to read and respond to pop-ups or other messages on the screen. |
|  | | Every window, object, and control has clearly named labels. Error messages appear close to the input.  We've ensured that the user interface and task flow are similar across different functions  Touch elements are the size of an average person's finger with adequate spacing. Users can increase the font size of the text on the page without loss of content or functionality. |

Table - Accessibility

## Gestures

Gestures are actions that represent inputs captured from users. Typically, Flutter supports the following gestures:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tap | Double-Tap | Long Press | Vertical Drag | Horizontal Drag |
| Tap events occur when a user touches the screen one time. | Double-tap events occur when a user touches the screen twice in quick succession in the same place. | Long press events are recorded when a user contacts the screen for a long time. | Vertical drag events are recorded when a user contacts the screen and moves the finger in a vertical direction without removing the finger. | Horizontal drag event is recorded when a user contacts the screen and drags their finger in a horizontal direction without removing their finger. |
| onTapDown  onTapUp  onTap  onTapCancel | onDoubleTap  onDoubleTapDown  onDoubleTapCancel | onLongPressDown  onLongPressStart  onLongPress  onLongPressMoveUpdate  onLongPressEnd  onLongPressUp  onLongPressCancel | onVerticalDragStart  onVerticalDragUpdate  onVerticalDragEnd | onHorizontalDragStart  onHorizontalDragUpdate  onHorizontalDragEnd |

Table - Gestures

## Semantics

According to [Attila Vágó](https://attilavago.medium.com/?source=post_page-----ecf547cd71ad--------------------------------), the Semantics widget allows developer to easily enhance their applications with accessible tagged featured by integrating the semantics widget into their existing projects. In the following examples, you can see how this is accomplished. The Semantics widget captures the upper most widget, but the excludeSemantics removes the remaining child items from being captures by a screen reader.

Text

Description automatically generated

Figure - Semantics Example

# Data and Backend

Today, the two most common storage solution include these solutions include Structured Query Language (SQL) or relational database solutions or not only SQL more commonly referred to as NoSQL

Table

Description automatically generated

Figure - NoSQL vs SQL from: Source: https://www.techtarget.com/searchdatamanagement/definition/NoSQL-Not-Only-SQLThere are a number of factors that are used in determining the selection of a storage system for this mobile application. The following criteria are important factors in selection a solution.

**Database selection criteria:**

1. **Structure** – offline, online, or both available
2. **Size** – The size of data refers to the quantity of the data you need to store and retrieve as critical application data. The amount of data may vary depending on a combination of the data structure selected, the ability of the database to differentiate data across multiple file systems and servers. So, you need to choose our database keeping in mind the overall volume of data generated by the application at any specific time and the size of data to be retrieved from the database.
3. **Speed/scale** – Speed and Scale address the time it takes to service incoming reads and writes to your application. Some databases are designed to optimize read-heavy apps, while others are designed to support write-heavy solutions. Selecting a database that can handle your app's I/O needs goes a long way to a scalable architecture.

For example, MongoDB is faster than MySQL when it comes to handling a large volume of unstructured data. On the other hand, MySQL would be faster for structured data.

1. **Data modeling** – Before you choose any database, it's advisable to perform Data Modelling. It is a representation of the data structures to be stored in the database and very powerful expression of the business requirements. Data Modelling is very helpful if your app contains features like search queries, reporting, location-based features, etc. Such apps require multiple databases to manage different types of data. For example, Uber uses MySQL, MongoDB and lot of other databases. They use MongoDB for their CDN and MySQL for business logic. Using MongoDB in their case made it very simple to store high volume incoming data.
2. **Data security** – When you're using synchronized and decentralized storage it is important to access, transmit, and store data securely. To cover this completely, you need to address authentication, data at rest, data in motion and read/write access.

Authentication should be flexible and allow for the use of standard, public, and custom authentication providers. Support for the anonymous access is also important for many apps. For data at rest on the server and client, you'll want support for both file system encryption and data-level encryption. For data in motion, communication should be over a secure channel like SSL or TLS. For data read/write access, the database should offer granular control over what data can be accessed and modified by users.

## SQLite Database

Based on the criteria listed above, the team decided to use a local instance of SQLite to persist the local caching of mail pieces and manage notifications for the application. This database is instantiated on first-use, which happens when the application first loads and requests the latest timestamp associated with a mail piece.

The schema for the database is as follows:

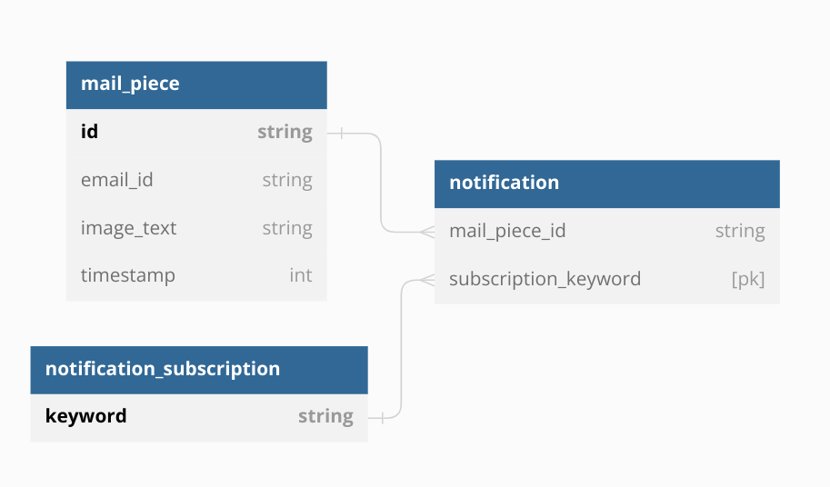


Figure - Database Schema

One important note about the use of SQLite, is the need to manually create migration logic when updating the schema. The current implementation uses the “version” field provided by the SQLite library to track the current iteration of the schema. In order to add, remove, or modify existing fields or tables:

* The developer must increment the version field.
* The developer must write migration logic in the setup function, gated by the new version number.
* The developer must validate that the changes are properly propagated.

## Notification and Notification Subscription Data Object – GitHub #412

In addition to caching the mail pieces in the local database cache, we also manage our notification subscriptions and their associated notifications. Notification subscriptions are added via the Manage page on the Notification view, which appends a new row to the “notification\_subscription” table. Removing a notification subscription on the Manage page results in the associated row in the “notification\_subscription” table being deleted. On subscription deletion, all associated notifications that are a result of that subscription are also removed via the cascading delete feature of SQL. Additionally, any time a mail piece is removed, the same will happen to any notifications that are associated with that mail piece.

# Firebase Analytics

Firebase is a great tool that functions as a backend-as-a-service tool that provides a lot of good out of the box functionality to its users, such as authentication, NoSQL database, analytics, and many more features. Firebase allows development teams to focus on business development and not to worry about the trickiness of backend coding. In this section we will go over the installation and configuration of Firebase Analytics. As requirements evolve, we will have to keep in mind that analytics data is only captured for a maximum period of 14 months. This may be a key factor in deciding whether to move to the data to a data warehouse because reviewing comparative periods is often an important analytical tool.

## Installation using Firebase CLI and Flutterfire

**Firebase CLI** (https://firebase.google.com/docs/cli#windows-standalone-binary)

1. Install NodeJS
2. Install Firebase CLI via the following command
   1. npm install -g firebase-tools
3. Login to Firebase
   1. firebase login
4. Test
   1. firebase projects:list

Flutterfire CLI

1. From the project directory, run the following command
2. Flutterfire configure

A screenshot of a computer

Description automatically generated

Figure - Flutterfire Configuration

## Setup / Configuration of Analytics Service

One the dependencies have been added to the project, there are a few additional components that need to be configured to ensure that objects within the project are appropriately tracked by Firebase.

Create the analytics\_service.dart file – this is the constructor class that is used by main() async in main.dart (shown below)

Figure - analytics\_service.dart

class AnalyticsService {  
  
 final FirebaseAnalytics \_analytics = FirebaseAnalytics.*instance*;  
  
 FirebaseAnalyticsObserver appAnalyticsObserver() =>  
 FirebaseAnalyticsObserver(analytics: \_analytics);  
  
 Future logScreens({@required String? name}) async {  
 await \_analytics.setCurrentScreen(screenName: name);  
 }  
  
}

Figure Main.dart

await Firebase.*initializeApp*(  
 options: DefaultFirebaseOptions.*currentPlatform*,  
);

Once those are created, the following code can be employed to track events, users (uspsMID), and page views.

* locator<AnalyticsService>().logScreens(name: "Digest Mail");

The following are each prefixed by **FirebaseAnalytics.instance.**

* setUserProperty(name: 'USPS\_Email\_MID', value: widget.mailPiece.uspsMID);
* logEvent(name: 'Link\_Navigated',parameters:{'itemId':text});

Sample Reports

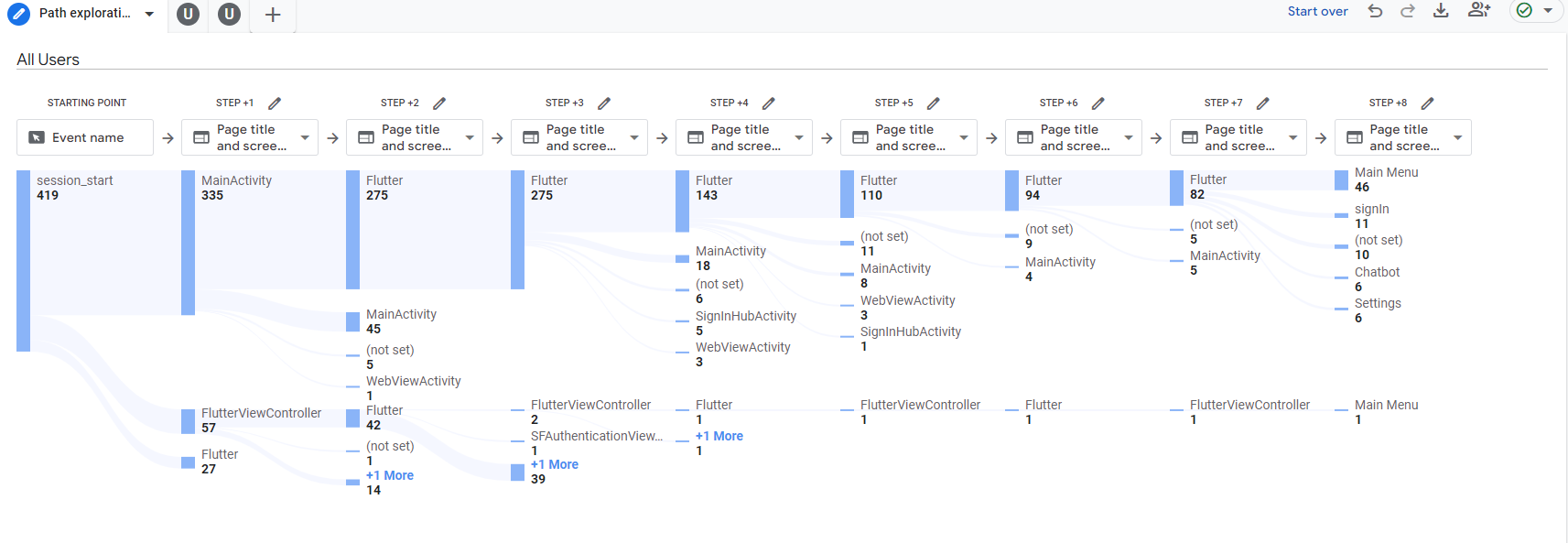
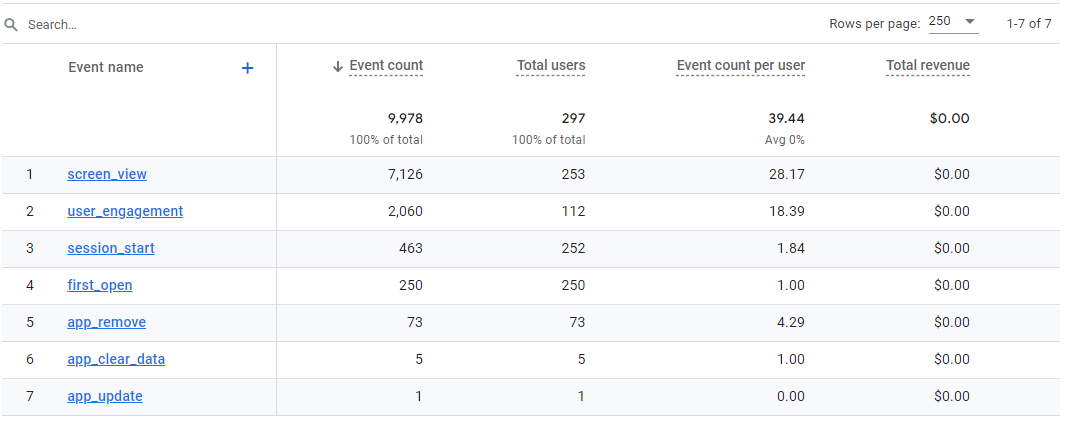


Figure 40 (Appendix A) Sample Google Analytics Path Exploration Report

Figure 41 (Appendix B) Sample Google Analytics Events Data Report

## Firebase Debug View

Android Studio combined with Google Firebase provides with ability to immediately visualize the impact of code changes using the Debug view

A screenshot of a computer

Description automatically generated with medium confidence

Figure 42 - Firebase Debug View

When running the project in Android Studio, the following command will allow you to test Google Firebase commands in real time. In the terminal, type in the following and press enter.

adb shell setprop debug.firebase.analytics.app [project name]

or com.umgc.summer2022 in our case

Then launch Google Firebase (<https://console.firebase.google.com/>) and connect to your project. When you use that command above, the Debug view will be available in the project, and you'll have access to real time statistics from the emulator.

Graphical user interface, application

Description automatically generated

Figure - Analytics Data

Examples of analytics data that is captured include the following:

* **Screen views** (any screen)
* **Search**
  + Keyword (mail sender and body)
* **Email view**
  + userProperty: uspsMID
  + Referring Links
  + Learn More
  + Set Reminders
* **Notifications**
  + Add Subscriptions
  + Remove Subscriptions
* **Settings**
  + Terms and Conditions (viewed)
  + Privacy Policy (viewed)

# Publishing

Publishing **MailSpeak** is an important step in making the application accessible to users. The Apple App Store and Google Play Store will be the main distribution channels for MailSpeak. In this section we will go over the process of how to publish in each store, where our Android application will be published in Google Play and our iOS application will be published in the Apple Appstore. These stores are how we manage the versions we release to the public and they are also a good public record of the updates we are releasing to the public.

## Google Play

Publishing the Android application is a multi-step process, where the team needs to check off to ensure the app will not be rejected by Google or any misconfigurations are released. **MailSpeak** is managed by Capstone teams across multiple semesters, so we will be utilizing the professor's information for the Google Play credentials. In this account there will be a Store Listing of the app, which will need to be updated whenever we want to release a new version.

Some of the prerequisites include removing debugging logs and replacing them with production logs. After that we need to build, test, and sign the release version of the application. Once the release version is prepared and ready to be distributed, an Android Studio Android Package Kit (APK) should be produced.

* Android Studio menu, select Build > Build Bundle/APK > Build APK.
* Once done, select the 'locate' button in the dialog.
* File explorer will open the location and that is the APK that will be used.
* It is good practice to rename the file with a version or build number.

Once on the store listing – there are a ton of promotional information that could be edited, such as promotional text, screenshots, icon, and more. Update this info as desired for your release. After that info is configured appropriately, on the left panel select the 'App Releases' section and there you will find three options: Production, Beta, and Alpha. We will be focusing on the Production option. Choose the release version 'Production', and then click 'Edit Release'. Upload the APK that we created in the previous step, add a realease name (generally version or build number), and add what's new about this release. Then submit the app for review. Google will review the application and we should hear back in up to 48 hours whether the app was approved or rejected. If rejected, review the reasons, adjust accordingly and follow these steps again. Google will also review any Google Assistant integration separately, this will not slow the regular review down, but Google Assistant will not work for regular users until it is approved. This usually takes 2-4 weeks.

## Apple Appstore

Publishing the iOS application is a multi-step process, where the team needs to check off to ensure the app will not be rejected by Apple or any misconfigurations are released. **MailSpeak** is managed by Capstone teams across multiple semesters, so we will be utilizing the professor's information for the Apple Appstore Connect credentials.

In XCode open up the **MailSpeak** workspace. Under the devices section, make sure to select 'Any iOS device', otherwise the following steps will not work. In the menu tab select 'Product' and then 'Archive'. XCode will then begin to build the app, if there are any errors, please fix them, otherwise follow along the prompts and answer the questions. Allow the signing to be done automatically with your certificates (\*) and then publish your app to App Store Connect. This process will upload your project to App Store Connect.

In the Appstore Connect account there will be a 'My Apps' section, which will hold the listing of **MailSpeak**. Go within **MailSpeak** and the 'App Store' tab. On the left there will be iOS App next to a '+' sign. Select the '+" sign and add store version, and select the build number that you just uploaded from XCode. Add a description of what is new in the build and select submit for review. This process will not take longer than 48 hours for Apple to approve or reject the build. If rejected, review the reasons, adjust accordingly and follow these steps again.

(\*) if you cannot automatically sign your app, make sure your developer workspace has the appropriate permissions and capabilities. This could be done in App Store Connect. https://support.appmachine.com/support/solutions/articles/80000978467

# Appendix

## Credits

Thank you to Professor Assadullah, Robert Wilson, Roy Gordon and Bob Dixon for providing us with this project and providing assistance when needed. This document was developed in tandem by.

Mike Conatser

Christopher Thorn

Kuleni Digga

Natan Tafese

Fahed Masood

Scott Huber

Harsh Gadani

Jahan Brahmabhatt

Ali Fahimi

Imoh Noah

Jack Shira

## Credit to Previous Cohorts

Thank you to the Summer 2022 class of SWEN 670. This application is based off of the original application that they developed. Some portions of this Programmers guide are based on their programmers guide due to this application being based on the one they developed. A reference to that document is listed in the references section above.

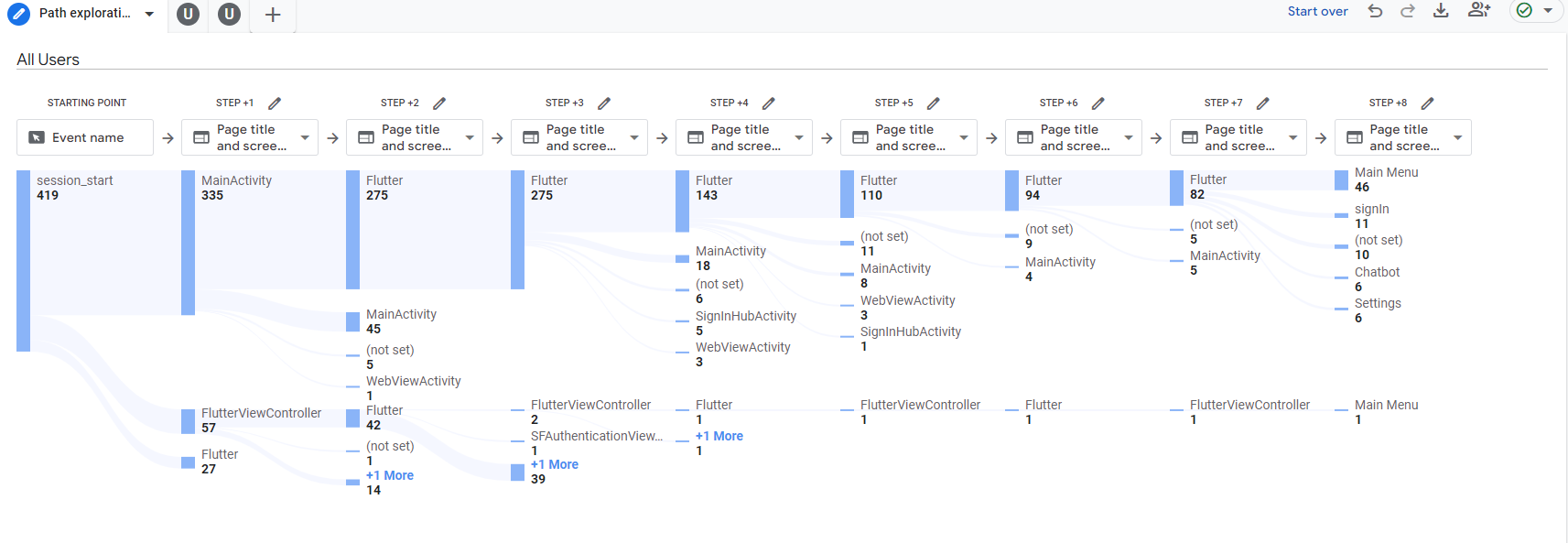
****

Figure 44 - Google Analytics - Sample Path Exploration Report

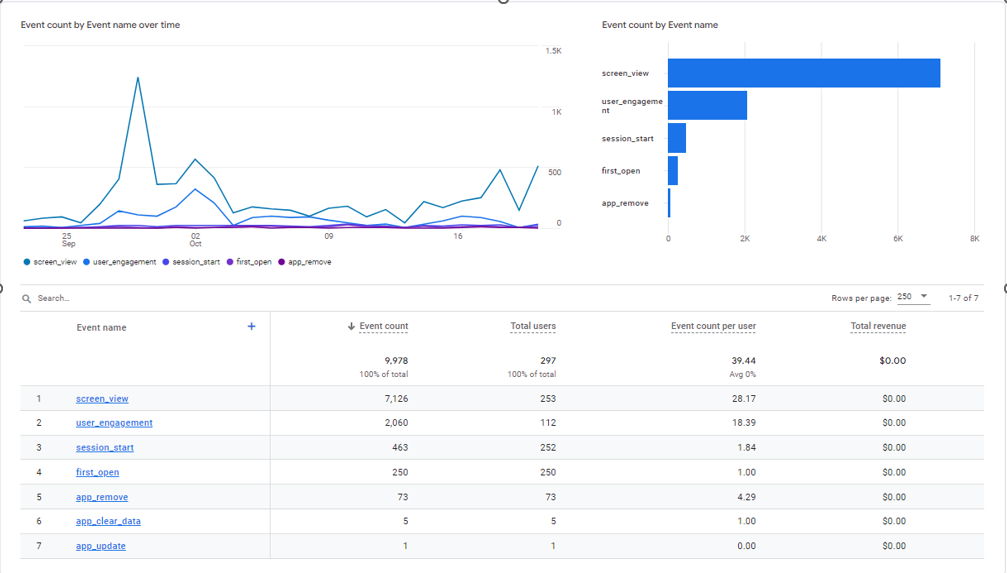


Figure 45 - Google Analytics – Sample Event Report