Research Paper Draft

Mobile Application AI Form Scriber

Milestone 4

SWEN 670

April 4, 2021

University of Maryland Global Campus

Presented by: Sylvia Lopez, Bertina Lee, Brian Malott, Karim Mansour, Komi Noukafou, Sompon Boontho, Joselito Ocampo, Ermias Seyoum, Arnaud Tako, Alex Dattilio, Vincent Leung, Ivy Pham, Roy Gordon, Johnny Lockhart

Table of Contents

[Abstract 4](#_Toc68560401)

[Introduction 5](#_Toc68560402)

[Agile Methodology and Scrum Framework 5](#_Toc68560403)

[Form Scriber application interaction with the SME 5](#_Toc68560404)

[Project Tools 7](#_Toc68560405)

[Architecture 8](#_Toc68560406)

[DevSecOps 9](#_Toc68560407)

[Software Application Development 11](#_Toc68560408)

[Login page 11](#_Toc68560409)

[Implementation 11](#_Toc68560410)

[**Challenges** 11](#_Toc68560411)

[Logout function 12](#_Toc68560412)

[Implementation 12](#_Toc68560413)

[Challenges 13](#_Toc68560414)

[Sliding menu 13](#_Toc68560415)

[Implementation 13](#_Toc68560416)

[Challenges 13](#_Toc68560417)

["Home" page 13](#_Toc68560418)

[Implementation 13](#_Toc68560419)

[Challenges 13](#_Toc68560420)

["Help" page 13](#_Toc68560421)

[Implementation 13](#_Toc68560422)

[Challenges 14](#_Toc68560423)

["Select Form Template" page 14](#_Toc68560424)

[Implementation 14](#_Toc68560425)

[Challenges 14](#_Toc68560426)

["Conversation" page 14](#_Toc68560427)

[Implementation 14](#_Toc68560428)

[Challenges 14](#_Toc68560429)

["View Reports" page 15](#_Toc68560430)

[Implementation 15](#_Toc68560431)

[Challenges 15](#_Toc68560432)

["Settings" page 15](#_Toc68560433)

[Implementation 15](#_Toc68560434)

[Challenges 15](#_Toc68560435)

[Business Logic 16](#_Toc68560436)

[Challenges 16](#_Toc68560437)

[Software Testing 16](#_Toc68560438)

[*Testing Tools* 16](#_Toc68560439)

[*Use Case* 16](#_Toc68560440)

[*Test Items* 17](#_Toc68560441)

[*Mobile Testing* 22](#_Toc68560442)

[*Advantages found while testing the UI with emulators* 22](#_Toc68560443)

[Conclusion 22](#_Toc68560444)

[References 24](#_Toc68560445)

# Abstract

Public service professionals now more than ever require all the assistance they can get to do what is needed from them. A team has been put together consisting of DialogFlow, Mobile, and DevSecOps to collaborate and create an Artificial Intelligence (AI) software application to assist professionals in public service. Currently, no AI software application using chatbot has been designed to prepare a text report for public service professionals required to document a portion of their job. This AI software will prepare a text report and access it through any mobile device for Android and iOS. This paper will explain in detail the steps that were taken to create and launch the AI application. From the research done by consulting with Subject Matter Experts (SME), the team was able to design an AI application with a robust architecture, using the help of Google Cloud Platform and its AI tools.

# Introduction

According to the United Nations Department of Economic and Social Affairs, public service professionals play a significant role in the Covid-19 pandemic, from the effects of an overwhelmed healthcare system to managing the socioeconomic impact (Kauzya & Niland, 2020). Public service professionals would benefit by allowing Artificial Intelligence technology to adjust the way they offer their services. In the case of our Form Scriber AI application, no other software using chatbot has been created to prepare a text report for public service professionals required to document a portion of their job.

Form Scriber uses Artificial Intelligence that, according to IBM Cloud Technology, "allows a computer to mimic the human brain" (IBM Cloud Education, 2020). Using Google Cloud Platform AI tools, the mobile application will listen to the conversation between the public service professional (interviewer) and the interviewee, then prepare notes that the customer will then edit before saving it. A tool will be used to guide the public service professional through viewing the end report. A template will be created that can be used later during the customer conversation with the interviewee. The template will have multiple elements that the AI part of the mobile application will fill in the required information as the conversation progresses.

This paper will cover how the Form Scriber application was created and what the team did to complete it. An SME that works as a Healthcare Medical Administrator for a Naval Hospital was consulted about the application. The Agile section of the paper will discuss the methodology used and the framework chosen for the project.

# Agile Methodology and Scrum Framework

Execution of the project using Agile with a Scrum approach was used through the development of the application. Scrum Artifacts that were used included a Product Backlog, Sprint Backlog, and the Increment (Scrum Guides, 2020). A Product Backlog was created with User Stories to ensure the product met expectations. The Project Manager, Lead Developer, and Senior Developer met with the DialogFlow Team members to determine what items of work on the Product Backlog would be changed. Based on the direction the application would go for the long-term objective. The [Architecture](#_Architecture) section of this paper was the outcome of the meeting.

Github was used to manage the Sprint Backlog work, and inside Github, a Kanban Board was created to address the issues that were being worked on. This allowed collaboration between all Mobile Application Team members, identifying the team member's name that pulled the issues and their assigned task.

Scrum Events consisted of the Sprint, Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective (Scrum Guides, 2020). Six Sprints were held during the Execution Phase of the project. The mobile team all worked together to discuss Sprint Planning and the most complex and important issues to approach from the Product Backlog to set a goal for the project completion. During the Sprint Retrospective, the team met to discuss how we could learn from the past week Sprints and plan for the next Sprint. By Sprint 4, work had been completed for specific issues, and all tasks were assigned to developers.

# Form Scriber application interaction with the SME

A Subject Matter Expert (SME) was consulted and agreed to meet to discuss the Use Cases and was shown mock-ups of the application. The SME that was consulted is an expert Health Care Medical Administrator for a Naval Hospital, having served in the US Navy for over twenty years and working in a hospital setting. Midway through the project, Stakeholders met to discuss changes that would move the project in a better direction. Finding a way that DialogFlow could use Google Docs to accomplish work that would advance the application. The SME agreed to meet to view changes in the application direction and advise the team if it was feasible and accommodating.

Questions for the SME:

1. Are you comfortable using your Google credentials to sign in to the mobile application?

User Response: If it is a work assigned Gmail account with a security assessment done, in agreement with our cybersecurity department, then I would be comfortable.

1. How do you like the menu options feature of sliding right to left for clicking on the menu?

User Response: It is a nice feature, but I would not have been aware of it if you had not shown me.

1. For the menu, would you prefer to have small icons appear on the side that you can slide out or just an icon to tap to open the menu?

User Response: Its good I like how everything is there, easy to navigate to what I need. Icon is convenient and easy to click on.

1. Once logged into the application, you are given a list of menu items to choose from. One of those menu items is Begin Conversation Session. The system will send a command request to initiate a form such as "Patient Registration." To begin the recording, you will click on the microphone button. Once you are done, you will click the microphone button to stop the recording. Is this process of events shown to you with our mock-ups offer you easy-to-follow navigation?

User Response: Yes, it looks great! Easy to navigate.

1. What features would you like included on the Homepage?

User Response: I can’t think of anything right now easy to view reports, settings self-explanatory. Begin conversation seemed easy to understand that that would take her to the dialog.

1. Once you’re done with the conversation, and you click on the record button to stop the recording, your given a URL link for your Google Docs, you will be able to view your reports as a pdf on your Google Docs. Is this something that you’re comfortable with?

User Response: She paused for a moment to try and understand the process. Then her reply was that it was useful to click on the record make your dictation and modify if needed

1. Going through the application mock-ups that you have seen what do you like most about it?

User Response: She paused to look and it, and said it looked good and could not think of anything else.

1. What did you like the least?

User response: I can't think of anything, I wonder if you could type into the app conversation, in case you're having a bad connection with the internet. Otherwise, I can’t think of anything to change.

1. Which parts of the software met your expectations?

User Response: great application easy to use.

1. Which parts of the software were difficult to use?

User Response: Again, this is pretty easy to use, and straight forward.

1. How do you plan to use the app?

User Response: Any doctor, physician assistant, nurse, could use this in their dictation, making their reports, their encounters, they can record their session verbally would be beneficial.

1. How would you rate your level of confidence using this app?

User Response: My confidence is still high; I look forward to seeing it in the near future and compete with other form scribing applications.

# Project Tools

The Form Scriber mobile application developer team used Android Studio as the IDE for development, editing, and compiling the source code. Flutter, a Google UI toolkit for developing natively compiled mobile applications with a single codebase, was used to create the cross-platform application. An object-oriented programming language Dart was used to program for both Android and iOS. The team used the Github repository to organize the project for code version control and collaboration between the team, along with Azure DevOps for automated builds and continuous integration, and SonarCloud for code quality and code analysis checks. The Google Cloud Platform, an online Platform-As-A-Service providing integration with all of Google’s service APIs, was used to integrate the Google Cloud IAM Oauth2 service and Firebase for application authentication and user authorization. Firebase Test Lab was used for automated unit tests and integration tests. The Google Play Developer Console was used for internal release testing and publishing the application to the Google Play store for alpha testing.

# Architecture

The goal of the Mobile team was to create a mobile application that would allow a user to fill out a profession form through passive voice recording processed through a back-end Natural Language Processor, DialogFlow. Existing software solutions were harnessed to develop the Form Scriber mobile application backend. Google Drive was used as an online document management system for form template storage and retrieval, "Form Scriber report" creation, modification, and recovery. Each user in this system must have their own Google account to adhere to Google's Oauth2 authorization flow for accessing users' documents through the Google Drive API. Users must also be a user within a Firebase Realtime Database.

The mobile application begins the process of authenticating a user with a Google sign-in on the "Log-in" page. Once the user is authenticated, a secondary authentication of the user will occur with a Google Firebase Realtime Database. This secondary authentication allows for the storage of refresh tokens for the Google Drive API access. It also allows administrators to manage users and/or groups with Firebase, Google Cloud IAM, and manage access control to Google Drive. Once a user has been authenticated with Firebase, the "Home" page will be displayed to the user.

The "Home" page offers navigation buttons to the various screens and functionality available throughout the mobile application. This includes the function of the "Form Scriber" process. This process begins when the user clicks the "Form Scribe a form" button on the "Home" page or through the slide menu. This will open the "Form template selection" page, which allows the user to select a form template document to use in creating a "Form Scriber report." These selectable form templates will only be those that the user is authorized to copy within the "Form template" directory in Google Drive. Once a template has been selected, the user must enter the name of the "Form Scriber report" that will be made from this template. If that "Form Scriber report" already exists in the designated "Form Scriber reports" directory, an error message will be displayed; otherwise, the template is duplicated to protect against overwrites and saved into the "Form Scriber reports" directory. The URL of this new "Form Scriber report" will then be sent to the "Report capture" page.

The "Report capture" page will send the provided document's URL to the DialogFlow agent associated with the Google Cloud project. This agent will use Google Drive to search through the document for fields represented by text within curly brackets, and create DialogFlow intents that will match the user's audio capture of the form fields. The user will then use voice capture and/or a manual input text box to capture each form field individually and see the response returned from DialogFlow's processing of the captured data. DialogFlow will record each captured field value to the document it was provided. A button at the bottom of the "Report capture" page is labeled "Complete Form Scriber report." If the user attempts to leave the "Report capture" page without clicking this button, a "Leave screen" confirmation dialog will appear. If the user confirms that they wish to leave, the newly-created "Form Scriber report" will be deleted from the "Form Scriber reports" directory. If the user presses the "Complete Form Scriber report" button at the bottom of the "Report capture" page, the user is sent to the "View reports" window to see their new report highlighted in the report list.

The user can reference completed reports in the "View Report" window, but they must open Google drive to edit or delete a completed "Form Scriber" report.  Navigating to the "Help" page, the user can review help information by clicking the "Help" button, and a new window will open with essential help articles. Navigating to the "Application settings" page, the user can review and edit the app's aesthetics, such as color and text size. They can also set the “Form template” and “Form Scriber report” directories that will be used in the "Form Scriber" process.



# DevSecOps

Within the last decade, the landscape of software development has changed drastically in the face of increasing customer expectations, increasing competition, and emergent processes and technologies that have driven the necessity for innovation. To meet the demanding pace of modern software engineering, organizations have had to adopt DevOps processes or development operations to ensure projects' success. This is the combination of development and IT operations teams aligning in principles and toolchains to create a streamlined and automated way of creating and releasing software. Leading the trends for 2020, continued automation and embedded security are paving the way for continual improvement and innovation in the DevOps industry space (ImpactQA, 2019). We aim to explore and conduct analysis on the effects of implementing such trends through the lens of a non-production development setting, such as present in an academic learning environment via a practicum, in order to derive insight into the feasibility of similar future efforts and to provide a data point for future research.

The multiple components and teams of the Form Scriber project posed an ideal environment in which a DevSecOps (DSO) team could be made into a proof of concept in order to facilitate coordinating the integration and deployment of said components into a cohesive end product. Therefore, the Form Scriber DevSecOps project was a complement to the Agile development methodology, providing services and technologies to development teams to enable rapid delivery of high-quality software on a frequent basis. The DSO team's goal was to enable the development teams to utilize a set of best practices and automation tools to build, test, secure, and deploy their services and applications. A continuous integration (CI) pipeline with a centralized code repository was provided to the development teams to enable automation and provide static code analysis to verify standards and security. A continuous delivery (CD) pipeline was also conceptualized to build and deploy.

The foundation of the DevSecOps infrastructure consisted of the source code repository. GitHub was selected as the tool of choice. This enabled the Mobile team of Dialogflow to collaborate on code creation and review. The mobile team was able to take advantage of the GitHub Projects feature to create a Kanban-inspired project board, which was used to plan and track tasks, features, code changes, and other project development activities. This allowed the team to collaborate more closely and see the project's status in terms of items to do, items in progress, items to review and approve, items to test, and items completed. This was crucial in tracing the course of development to attribute specific items to team members in order to manage and assign work items. An Activity menu kept track of contributions and changes by team members. The GitHub Issues feature integrated with Projects and was utilized to track project and development issues to resolve one single list. A simple Open or Closed status for the items provided an easy way to make a checklist for project tasks. As the Form Scriber project operated on no budget, these project management features filled the gap and ultimately allowed the team to set accountability and meet expectations required in an industry project.

The team took advantage of the GitHub repository core features, such as using git, a decentralized source control method consisting of local and remote repositories. The team created individual feature branches representing the feature additions and changes to the code, which then integrated with the main integration branch. As a customization request by the development team, this branch was made the default branch for the project. Individual contributor developers created code in the feature branches then created pull requests to merge into the development branch. Pull requests allowed the team to utilize a standard process to track and merge code while keeping traceability and maintainability high. The merge would be reviewed and approved by at least one other team member, and the merge was only able to be done by the designated developer lead. This branch was merged into the main branch that represented the production version of the code. Team members would create a pull request for this, which a DevSecOps team member would review for compliance to code standards and then merge.

Using pull requests, the team was able to see in real-time automated scans and checks that were conducted by SonarCloud for the code being pushed. Security scans provided feedback regarding compliance to coding best practices as well as security best practices. Any bugs or vulnerabilities were identified and flagged upon the scan after the pull request, and the status was updated immediately to the repository. Major issues identified did not allow for the merge to take place and required DSO team member review and/or remediation by the development team. This ensured a higher-quality end product and provided a real-time learning experience for some team members. If any issues were identified, any team member could browse the issue details in the SonarCloud page and review recommendations for remediation. After changes were pushed, scans were re-run automatically and the status updated.

The code quality checks were only one method to ensure high-quality code. Other features enabled by the DevSecOps infrastructure relied on project management intervention to ensure a consistent development culture that relied on cooperation by all team members to lend insight into other team members' code. This was done by conducting peer code reviews upon pull requests, which allowed other developers to see others' work and approve or provide suggestions and recommendations. This also provided a means to make more secure code as DevSecOps provided project management and development teams guidelines by which to implement security best practices via an appropriate security-centric mindset and continued learning.

Lastly, the Mobile development team was able to take advantage of a continuous integration pipeline created in Azure DevOps to automatically build, test, and monitor the status of the codebase. The pipeline integrated with GitHub and provided tasks by which to pull the code from the repository, install dependencies, and execute application build and test commands. This was done on each pull request and merge in the main repository branch. Within minutes, team members would be automatically notified with the status of the build, allowing for a fast feedback cycle. Email notifications for build success or failure was sent to a central DevSecOps email account, which integrated with a Microsoft Teams channel that was visible to all project stakeholders. For outstanding issues, DSO team members notified the Mobile team members to advise on fixes to prevent stale issues from becoming major problems. The build pipeline made use of plugins and extensions specific to the Mobile team project for Dart/Flutter projects that could build and test such code not native to the Azure DevOps platform. Having an automated solution to build and test code from multiple contributors allow the team to have confidence that they were not pushing broken code and that potential issues related to environment dependencies could be more removed.

Within the given short timeframe, a parallel research and development effort into a DevSecOps framework, Advance Development Factory (ADF), was completed and will inform future development cycles and provide additional benefits, including for the Form Scriber Mobile component. This framework could be customized and integrated with the Mobile component to provide additional facilitation for removal of dependencies to better ensure successful builds via Docker containers. A deployment pipeline for ADF was also realized that provided a working template by which future DevOps efforts could incorporate into development team projects for standalone deployment pipelines. The DevSecOps integration allowed for the Mobile team to harness industry-accepted practices related to IT infrastructure, Agile methodology alignment, and take advantage of current trends in software development. This provided invaluable feedback to development team members and stakeholders on the status and health of the development lifecycle during every step of the effort.

# Software Application Development

## Login page

### Implementation

Two developers were paired to research and develop the UI for this feature. The Login page is the initial page a user interacts with to gain access to the Form Scriber app, incorporating authentication through Firebase and Google sign-in. The Login feature took extensive time due to figuring out the best approach to accessing Google credentials and whether to include Firebase. The back-end to make authentication work was created by two other mobile team developers. The login process involves the following steps:

1. A user logs in to the Form Scriber application with a Google account.
2. Google returns a unique token used to authenticate and identify our account for all future actions with Google.
3. The Form Scriber application attempts to authenticate the user with Firebase, passing the provided Google authentication ID token and access token.
4. Upon successfully authenticating the user with Firebase, the Form Scriber application opens its "Home" page, and the user's login is monitored and managed through the "FirebaseAuth" API.

### **Challenges**

There were significant challenges in implementing this functionality. The initial estimated work hours for completion were far exceeded by the developers tasked with this work item. The main factor contributing to the developer’s difficulty was onboarding the developers from a previous role on the former “Web Services” team to a new role on the Mobile team. There was a learning curve associated with first-time use of Android Studio, along with extended research into Firebase and Google Sign-In. Although Google provides extensive documentation on their APIs, there was not quite much clear documentation on how to implement Firebase and Google Sign-In in a Flutter project. The work item had to be swarmed by two more developers prior to the Milestone 3 deadline in order to get a working prototype. The Google Sign-In functionality was finally achieved by following these steps:

1. Created a new GCP project from scratch. called "formscriber2"
2. Configured app credentials in GCP console project.
3. Configured OAuth consent screen to be able to register the app with Google for Drive API access.
4. Specified the Publishing status as “Testing”. This allows us to access Google APIs for interaction with up to 100 Test Users and a max of 10000 Token grants per day.
5. Created a new Oauth2 client credential with application type “Android” in GCP console.
6. Created a new Firebase project in GCP console linked to GCP project of the same name.
7. Enabled Google Sign In authentication method in Firebase project.
8. Imported google\_sign\_in, firebase\_auth, firebase\_core, and googleapis flutter packages into the flutter project’s pubspec.yaml dependencies.
9. Configured android/build.gradle and android/app/build.gradle files for connecting to Google-Services API and Firebase APIs specifically from android devices.
10. Extracted the local debug Private Key from AndroidDebugKey keystore using android Gradle Daemon in order to allow Firebase authentication from debugger builds running locally within Android studio.
11. This required a temporary change to the formbot\_app/android/gradle.properties file: change property “org.gradle.jvmarg” from “-Xmx1536M” to “-XX:MaxHeapSize=256m -Xmx256m”
12. Re-sync the android gradle settings.
13. Run the command “gradlew signingReport” from the formbot\_app\android directory.
14. Add the SHA1 and SHA-256 fingerprints of this “AndroidDebugKey” to the Firebase project settings
15. Change gradle.properties property “org.gradle.jvmarg” back to “-Xmx1536M”
16. Generated a new google-services.json client token from Firebase project settings for importing into the android module of our flutter project
17. Ran “flutter clean”, then ran “flutter pub get” and re-ran the application.
18. Created a release keystore locally for application signing required before upload to Google Play.
19. Extracted SHA1 and SHA-256 fingerprints of the release keystore and added them into the Firebase project settings.
20. Generated a new google-services.json client token from Firebase project settings for importing into the android module of our flutter project
21. Ran “flutter clean”, then ran “flutter pub get” and re-ran the application.

Reference: <https://www.seemuapps.com/swift-firebase-google-login-tutorial>

## Logout function

### Implementation

The "Logout" function was created to allow users to sign out of the Form Scriber app and disconnect their accounts from the app. A "Logout" button was implemented on the "Home" page and the slide menu to complete this function. This button had an "onClickListener" event attached to it, and the "onClick" method would call the "logOut" method. Upon a successful logout, the user is returned to the "Log-in" page. The following reference link was used as an example to implement this feature: <https://developers.google.com/identity/sign-in/android/disconnect>

### Challenges

The same challenges which applied to the Login function also applied to the Logout function.

## Sliding menu

### Implementation

The original requirements included a sliding menu exposed by the user dragging their thumb from right to left. A proof of concept was created using the built-in drawer widget from Flutter. This widget places an icon at the top of the screen, whereas the user can tap on the icon to bring up the menu. The menu can also be exposed by sliding the thumb from right to left. We found a tutorial to create a more elaborate sliding menu, however, due to time constraints, this required a lot more programming and troubleshooting than the timeline of the project allowed (Collapsible Sidebar and Navigation Drawer, 2019).

For the menu to open from right to left, the Flutter widget "endDrawer" was used. The "endDrawer"positions the menu to open from right to left. If the desire was for the menu to open from left to right, the "drawer" would be used. Since this menu would be standard among all pages of the application, the menu was placed in its file under the "widgets" folder; therefore, any changes needed to the menu will be made in one place and accessible by all pages.

The list items in the sliding menu were placed into their own method in the "app\_drawer.dart" file to support maintainability. This allows for additional menu items to be implemented in the future while maintaining the consistency of the expected formatting. The "\_createDrawerItem" method takes the name of the desired icon for the menu item, text of the menu item, and the route, or link, to the page that the menu item is associated to. All formatting remains within the "\_createDrawerItem" method, and the focus is on the desired contents.

### Challenges

None

## "Home" page

### Implementation

The "Home" page provides users with a familiar and centralized navigation hub for all of the mobile applications' functionality. A warning message is displayed at the bottom of the "Home" page for the user to read before proceeding with any action. The home page was designed as a StatelessWidget because its content will not be changing over time. Flutter's user interface recommendations of inline styling were also followed in creating the "Home" page.

### Challenges

None

## "Help" page

### Implementation

The "Help" page allows users to review guides for the Form Scriber application. It becomes accessible once the user has logged in to the Form Scriber application. The user can open the "Help" page by clicking the "Help" button on the "Home" page or in the slide menu. Help articles on this page include topics such as logging-in to the application, navigating the "FormScribe" process, and troubleshooting. The "help" page is a StatelessWidget.

### Challenges

None

## "Select Form Template" page

### Implementation

This is the initial page of the "Conversation" process, which precedes the conversation. The UI of this page was developed using widgets to create the drop-down button and display the list of form templates available to the user. The user selects a template by clicking on a form. The page also includes a textbox field where the user will type a name for the report. A button was created for the user to click on "Proceed" The links below were used to access codes for examples of creating the drop-down list in Flutter. There was difficulty with testing the code due to problems accessing the emulator. This was later resolved. The backend developers needed to set up Google API to load the form templates into the drop-down list.

<https://yashodgayashan.medium.com/flutter-dropdown-button-widget-469794c886d0>

<https://www.developerlibs.com/2019/09/flutter-drop-down-menu-list-example.html>

### Challenges

None

## "Conversation" page

### Implementation

The “Conversation” page gets shown after a user has selected a form template from the "Select Form Template" page. The page is controlled by a Stateful Widget called “Chat” which implements several flutter packages to establish communication with and send and receive messages to and from the DialogFlow agent and stream audio input to the General Remote Procedure Call API provided by Google. The package “dialogflow\_grpc” is used to accomplish both of these tasks in conjunction with the “sound\_stream” package. A secondary Stateful Widget called “App\_Body” is used as a UI structure for presenting data received from the DialogFlow agent on screen.

### Challenges

The major challenges surrounding the Conversation screen revolved around selecting a full featured flutter package available for DialogFlow integration. Upon initial research, the choice seemed to be clear, using the “flutter\_dialogflow” package. However, this proved to be a non-starter because that package was outdated and contained dependencies on old versions of core packages which conflict with the recent dependencies of the google\_sign\_in and firebase\_auth packages. The next choice was “dialog\_flowtter” which had updated dependencies, but was missing a key method which is StreamingDetectIntent. This method is required for sending audio streams to the DialogFlow agent. Further research resulted in the only available pagkage without conflicting cross-dependencies and with audio streaming capability being “dialogflow\_grpc”.

However, during development of a prototype using “dialogflow\_grpc”, it was discovered that even this package had a bug within the StreamingDetectIntent method. This bug presented itself by the behavior that when a local microphone listen event was triggered, the audio streams to the gRPC server, which forwards to the Dialogflow agent correctly, but the bidirectional stream which waits for a response from DialogFlow gets closed too early before receiving any actual intent responses from the agent.

To overcome this shortcoming, a breakdown of the protocol buffers used within the dialogflow\_grpc package was required. The final implementation uses the exact StreamController and Stream directly withing the Chat class, with synchronous methods to start and stop this Stream upon microphone events.

Resources: <https://pub.dev/packages/dialogflow_grpc> ,

<https://www.leeboonstra.dev/apis/dialogflow_flutter_grpc/> ,

<https://cloud.google.com/dialogflow/es/docs/reference/rpc/google.cloud.dialogflow.v2#google.cloud.dialogflow.v2.Sessions.StreamingDetectIntent> ,

<https://api.flutter.dev/flutter/dart-async/StreamController-class.html> ,

<https://pub.dev/packages/sound_stream/example>

## "View Reports" page

### Implementation

The "View Reports" page allows users to view existing reports. There exists a link to each conversation that has been previously completed. Form Scriber will send a request to Google Drive using the "googleapis 2.0.0" dart package to retrieve and display a list of all the reports the user has the authorization to read. The user will then select the report they wish to view. Form Scriber will use the associated report identification number to retrieve the report and display it on the screen for the user to view. The view reports widget is built as a StatefulWidget because it will change over time as more conversations have been completed. The user interface for this page has been inline styled per the Flutter framework.

### Challenges

None

## "Settings" page

### Implementation

The settings page allows the user to select the theme of the application, having the choice between a Light theme and a Dark theme. This page also allows the user to define the Google Drive URLs for their Form Template directory and their Form Scriber Reports directory. The code uses the built in Dart packages “dart.core” and “material.dart” to control application-wide colors and fonts for all widgets, and the Google Drive API modules within the “googleapis” package.

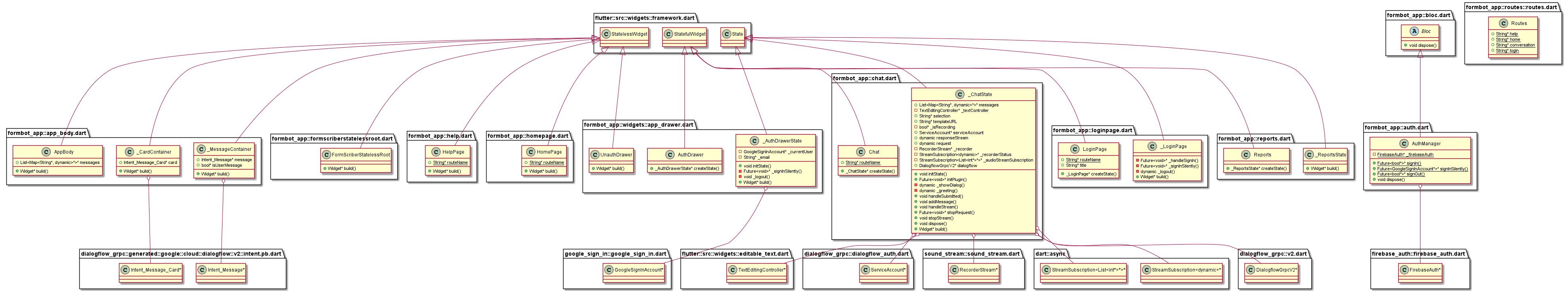
### Challenges

None

## Business Logic

The business logic provides the backbone functionality for the mobile app's UI and involves Google services. The login screen will use Flutter's "google-sign\_in" dart library to authenticate a user's credentials. The "Report capture" screen will use Flutter's "googleapis" library to create the duplicate template document and fetch existing document names and URLs for template selection from the user's Google Drive account. This screen will also communicate with an agent of Google's DialogFlow service through an embedded Google Assistant and a predefined action phrase.

Each UI screen has a StatefulWidget class which is the main data controller of that page and controls the logic for handling input data, UI widget states, external communication, and processing output in the form of a report. The UI conversation screen, settings screen, and report view screen all depend on their respective StatefulWidget class as a Business Logic Component for content in some form. Communication between the Mobile app and DialogFlow agent, and requests to the Google APIs are handled by StatefulWidgets along with Dart StreamController classes.



*Figure 1: UML Class Diagram of FormScriber application packages.*

### Challenges

None

# Software Testing

Software testing for this project focuses on unit testing, end-to-end testing, and integration tests. The end-to-end testing provides the UI testing capabilities to simulate the user interaction with the system. For instance, how the mobile application responds to the user requests and functionality of the AI while engaging in the conversation activities. The software testing is performed to ensure that the system has features and functions that meet the software requirements specifications and enable the end-user to accomplish their tasks.

## *Testing Tools*

The tools that we are using for testing include the following:

* Flutter testing
* Firebase Test Lab

## *Use Case*

All of the use cases listed below have been tested in Android Studio using automation testing to include unit testing, integration testing, and Firebase Test Lab.

1. Sliding menu
2. Log In
3. Log Out
4. Begin Conversation Session
5. Specify a From Template to Form Scriber
6. Specify "New Prescription" to Form Scriber
7. Specify "Medical History" to Form Scriber
8. Request Report
9. View Reports
10. View Settings

11. View Help.

## *Test Items*

The following tests were planned based on the Use Cases listed above.

Table 1 - test item is based on Use Case: Sliding Menu, the user swipes right to left to see the sliding menu options.

Table 1 – Sliding Menu

|  |  |  |  |
| --- | --- | --- | --- |
| **Items to be Tested** | **Version Number** | **Test Action** | **Expected Results** |
| 1. Sliding menu | 1.0 | 1. The actor swipes right to left | 2. The system displays the menu to allow the user to click on the menu item for the Homepage, Begin Conversation, View Reports, Settings, Help and Log Out |
|  |  |

Table 2 test item is based on Use Case: Login, the display and functionality of logging a user into the system.

Table 2 - Login

|  |  |  |  |
| --- | --- | --- | --- |
| **Item to be Tested** | **Version Number** | **Test Action** | **Expected Results** |
| 2. Login | 1.0 | 1. The actor taps on the login button on the landing screen. | The system will display the login screen with the Username and Password fields. |
| 2. 3. The actor enters their Google credentials, username and password, then taps the login button | The system will validate the credentials. |
|  |  |  | The system displays the main page. |

Table 2.1 Is continued from the test item shown in Table 2 Login.

Alternate paths:

* If the user enters an invalid username and/or password.

Table 2.1 – Continuation of Table 2 Login, Test Item with Alternative Path

|  |  |  |  |
| --- | --- | --- | --- |
| **Item to be Tested** | **Version Number** | **Test Action** | **Expected Results** |
| 2. Login | 1.0 | 1. The user enters an invalid username and/or password. | The system displays an error message indicating the username and/or password is invalid, the system remains on the login screen. |

Table 3 test item is based on the Use Case: Logout, the display and functionality of logging a user out of the system.

Table 3 - Logout

|  |  |  |  |
| --- | --- | --- | --- |
| **Item to be Tested** | **Version Number** | **Test Action** | **Expected Results** |
| 3. logout | 1.0 | 1. The actor taps on the logout button on any screen. | The system will end the session and displays a logout confirmation screen. |

Table 4 test item is based on the Use Case: Begin Conversation Session. Navigate to the "Form Scriber" chatbot activity to begin the conversation session. Connects to Form Scriber to initiate communication.

Table 4 - Begin Conversation Session

|  |  |  |  |
| --- | --- | --- | --- |
| **Item to be Tested** | **Version Number** | **Test Action** | **Expected Results** |
| 4. Begin conversation session. | 1.0 | 1. The actor taps the Begin Conversation button on the main screen | The system requests a list of available form templates through the Google Drive API, based on the Google Drive repository which the customer has preconfigured. This list populates a DropdownButton menu. The system navigates to the conversation screen and prompts the user to select a template from the drop down list before they can begin speaking. |
|  |  | 2. The actor taps on the drop-down menu. | The system reveals a list of available form templates. |

Table 5 test item is based on the Use Case: Specify "Patient Registration" to Form Scriber. Sends formName for "Patient Registration" to Form Scriber.

Table 5 - Specify a Form Template to Form Scriber

|  |  |  |  |
| --- | --- | --- | --- |
| **Item to be Tested** | **Version Number** | **Test Action** | **Expected Results** |
| 5. Specify a Form Template to Form Scriber. | 1.0 | 1.The actor selects the form template from the drop down list. | 1.The system displays the form name on screen and gets the URL in memory, stored as a String. |
|  |
|  |  | 2. The actor clicks "Proceed". | 1. The system sends the form URL corresponding to the selected form name to the DialogFlow agent and receives a response from DialogFlow indicating it's ready to listen for field input. |
|  |  | 3. The actor taps the microphone button. | 1. The system stops recording the audio. |
|  |  | 4. The actor speaks into their microphone, stating the name of the specific field they want to update. | 1.The system automatically  stops recording audio after a standard moment of silence of 2 seconds, and converts their audio into text and sends it to the DialogFlow agent. |
|  |  |  | 2. The DialogFlow agent responds with a confirmation message that the desired intent is ready to be recorded. |
|  |  | 5. The actor taps the microphone button and speaks the field entry detail to get written. | 1. The system converts audio and sends this message as a test to the DialogFlow agent, then receives a response from DialogFlow confirming the specific field was written. |

Table 6 test item is based on the Use Case: Specify "New Prescription" to Form Scriber. Sends formName for "New Prescription" to Form Scriber.

Table 6 - Specify "New Prescription" to Form Scriber

|  |  |  |  |
| --- | --- | --- | --- |
| **Item to be Tested** | **Version Number** | **Test Action** | **Expected Results** |
| 6. Specify "New Prescription" to Form Scriber | 1.0 | 1. The actor selects the form from the From Scriber chatbot suggestion. | The system sends the formName corresponding to New Prescription to the DialogFlow agent and receives a response from DialogFlow. |
|  |
|  |  | 1. The actor taps the microphone button | The system starts recording the audio. |
|  |  | 1. The actor taps the microphone button | The system stops recording the audio. |

Table 7 test item is based on the Use Case: Specify "Medical History" to Form Scriber, sends form ID for "Medical History" to Form Scriber.

Table 7 - Specify "Medical History" to Form Scriber

|  |  |  |  |
| --- | --- | --- | --- |
| **Item to be Tested** | **Version Number** | **Test Action** | **Expected Results** |
| 7. Specify "Medical History" to Form Scriber. | 1.0 | 1. The actor selects the form from the Form Scriber chatbot suggestion. | The system sends the formName corresponding to New Prescription to the DialogFlow agent and receives a response from DialogFlow. |
|  |
|  |
|  |
|  |  | 1. The actor taps the microphone button | The system starts recording the audio. |
|  |  | 1. The actor taps the microphone button | The system stops recording the audio. |

Table 8 test item is based on the Use Case: Request Report, sends form URL of the selected template to Form Scriber.

Table 8 - Request Report

|  |  |  |  |
| --- | --- | --- | --- |
| **Item to be Tested** | **Version Number** | **Test Action** | **Expected Results** |
| 8. Request Report | 1.0 | 1. The user provides the required data in the conversation in the chatbot.  2. The user will indication the request for completed report by saying "print". | The system will request the report URL from the DialogFlow agent. |
| The system will receive the report URL from DialogFlow. |

Table 9 test item is based on the Use Case: View Reports, displays report history.

Table 9 - View Reports

|  |  |  |  |
| --- | --- | --- | --- |
| **Item to be Tested** | **Version Number** | **Test Action** | **Expected Results** |
| 9. View Reports | 1.0 | 1. The actor taps on the View Reports link from the menu navigation. | The system requests a list of links to the customer's designated Google Drive document repository. The system displays the list of reports available to view. |

Table 10 test item is based on the Use Case: View Settings, displays and sets settings.

Table 10 - View Settings

|  |  |  |  |
| --- | --- | --- | --- |
| **Item to be Tested** | **Version Number** | **Test Action** | **Expected Results** |
| 10. View Setting | 1.0 | 1. The actor taps the Settings link in the menu navigation. | 1. The system displays the available settings on the Settings screen. |

Table 11 test item is based on the Use Case: View Help, displays application's help system.

Table 11 - View Help

|  |  |  |  |
| --- | --- | --- | --- |
| **Item to be Tested** | **Version Number** | **Test Action** | **Expected Results** |
| 11. View Help | 1.0 | 1. The actor taps the Help link on a screen in the system. | 1. The system displays the help screen. |

## ***Mobile Testing***

**Tests were run against the Form Scriber mobile application to ensure all functionality and usability complied with the stakeholder requirements before the launch date. The User Interface was tested using an Android and iOS emulator through Android Studio.**

## *Advantages found while testing the UI with emulators*

Due to the project budget and resources available, it was impossible to test with many actual devices. Because of this, UI testing was done with emulators. Emulators made it possible to test the resolution on various devices. Different operating systems for Android and iOS were tested using emulators. Testing was done by two software testing engineers who were able to test in parallel on the same device's emulators.

# Conclusion

Creating the Form Scriber AI mobile application through a chatbot was designed to leave behind documentation for a team to adjust the application to adapt to their organization requirements. Being able to offer public service professionals a tool that will listen to a conversation between the interviewer and the interviewee and then send the captured recording to the DialogFlow AI for processing the voice recording is a new technology that will be useful through years to come. More can be added to the application through time with advancements that AI will bring to the future of application development.

# References

Collapsible Sidebar and Navigagtion Drawer . (2019, February 10). *Flutter UI*. Retrieved from YouTube: https://www.youtube.com/watch?v=2SjvhAUR9aw

Flutter - drop down menu list with example. (2021, March 29). Retrieved from Developer Libs: https://www.developerlibs.com/2019/09/flutter-drop-down-menu-list-example.html

IBM Cloud Education. (2020, June 3). *Artificial Intelligence (AI)*. Retrieved from IBM: https://www.ibm.com/cloud/learn/what-is-artificial-intelligence

ImpactQA. (2019, November 15). *8 DevOps trends to watch for in 2020*. Retrieved from Dzone: https://dzone.com/articles/8-devops-trends-to-know-in-2020

Kauzya, J.-M., & Niland, E. (2020, June 11). *Department of Economic and Social Affairs Economic Analysis*. Retrieved from United Nations: https://www.un.org/development/desa/dpad/publication/un-desa-policy-brief-79-the-role-of-public-service-and-public-servants-during-the-covid-19-pandemic/

Perera, Y. (2020, April 7). *Flutter dropdown button widget*. Retrieved from Medium: https://yashodgayashan.medium.com/flutter-dropdown-button-widget-469794c886d0

Scrum Guides. (2020). *The 2020 Scrum Guide*. Retrieved from Scrum Guides: https://www.scrumguides.org/scrum-guide.html#increment

Signing out users and disconnecting accounts. (2021, March 29). Retrieved from Google Identity: https://developers.google.com/identity/sign-in/android/disconnect

Swift Firebase Google Login Tutorial. (2019, April 21). Retrieved from Seemu Apps: https://www.seemuapps.com/swift-firebase-google-login-tutorial