Programmer Guide

Form Scriber Mobile Application

Milestone 3 – Mobile Team

Software Engineering Project

SWEN 670

March 16, 2021

Version 1.0

Presented by: Sylvia Lopez-Willis, Sompon Boontho, Alex Dattilio, Bertina Lee, Brian Malott, Karim Mansour, Komi Noukafou, Joselitio Ocampo, Ermias Seyoum, Aruand Tako

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| DATE | VERSION | DESCRIPTION | AUTHOR |
| 03/16/2021 | 1.0 | Initial documents | Sylvia Lopez-Willis, Sompon Boontho, Alex Dattilio, Bertina Lee, Brian Malott, Karim Mansour, Komi Noukafou, Joselitio Ocampo, Ermias Seyoum, Aruand Tako |
|  |  |  |  |
|  |  |  |  |

Programmer Guide Approvals

|  |  |  |
| --- | --- | --- |
| Name | Signature | Date |
| Approved by:  "Software Lead" |  |  |
| Approved by:  "Project Manager" |  |  |
| Approved by:  “Stakeholder-Dr. Mir Assadullah” |  |  |

Table of Contents

[1 Introduction 5](#_Toc66805090)

[1.1 Purpose 5](#_Toc66805091)

[1.2 Intended Audience 5](#_Toc66805092)

[1.3 Technical Project Stakeholders 5](#_Toc66805093)

[1.4 Definitions, acronyms, and abbreviations 5](#_Toc66805094)

[1.5 References 6](#_Toc66805095)

[2 System Architecture 6](#_Toc66805096)

[2.1 Architectural Design 6](#_Toc66805097)

[3 Setting up the application 7](#_Toc66805098)

[3.1 Downloading Android Studio 7](#_Toc66805099)

[3.2 Downloading Flutter and Dart 17](#_Toc66805100)

[3.3 Frontend Development 23](#_Toc66805101)

[3.3.1 Landing Page 23](#_Toc66805102)

[3.3.2 Login page 23](#_Toc66805103)

[3.3.3 Home page 23](#_Toc66805104)

[3.3.4 Select Form Template page 23](#_Toc66805105)

[3.3.5 Record Form Fields page 24](#_Toc66805106)

[3.3.6 Settings page 24](#_Toc66805107)

[3.4 Backend Development 25](#_Toc66805108)

[3.5 Running Tests 26](#_Toc66805109)

[3.5.1 Android studio 27](#_Toc66805110)

[3.5.2 Test types and location 27](#_Toc66805111)

[3.5.3 Flutter Test Package 28](#_Toc66805112)

[3.6 Running end-to-end tests 29](#_Toc66805113)

[4 Code structure 31](#_Toc66805114)

[4.1 Frontend 31](#_Toc66805115)

[4.1.2 main.dart 33](#_Toc66805116)

[4.1.4 HomePage 37](#_Toc66805117)

[4.1.5 App\_drawer. Dart 40](#_Toc66805118)

[4.1.6 Help. Dart 41](#_Toc66805119)

[4.1.7 Conversation. Dart 42](#_Toc66805120)

[Appendices: 44](#_Toc66805121)

[Appendix A – References 44](#_Toc66805122)

# 1 Introduction

## 1.1 Purpose

The purpose of the Programmer’s Guide is to allow future software developers a means of knowing how the Form Scriber Mobile Application was developed. The developers can see the tools used, the language selected, and the application's overall architecture.

## 1.2 Intended Audience

The intended audience for this document will be Software Developers and Testers that wish to develop or maintain the application. This Programmer Guide assumes the software developers have no previous knowledge of the AI Form Scriber application using a chatbox and providing a means for public service professionals to prepare a text report.

## 1.3 Technical Project Stakeholders

The table below shows a list of the project stakeholders for the Form Scriber Mobile Application:

Table - Project Stakeholders

|  |  |
| --- | --- |
| Name | Role |
| Dr. Mir Assadullah | Stakeholder (Project Owner) |
| Sylvia Lopez-Willis | Project Manager |
| Karim Mansour | Lead Developer/Backup PM |
| Alex Dattilio | Developer |
| Bertina Lee | Developer |
| Brian Malott | Developer |
| Komi Noukafou | Developer |
| Joselito Ocampo | Developer |
| Ermias Seyoum | Developer |
| Arnaud Tako | Developer |
| Sompon Boontho | Software Tester |

## 1.4 Definitions, acronyms, and abbreviations

The table below shows the acronyms, abbreviations, and definitions of the corresponding row.

Table - Definitions, acronyms, and abbreviations

|  |  |
| --- | --- |
| Acronyms and Abbreviations | Definitions |
| AI | Artificial Intelligence |
| JVM | Java Virtual Machine |

## 1.5 References

Table - References

|  |  |
| --- | --- |
| Title | Reference |
| Test your app | [https://developer.android.com/studio/testevelopers](https://developer.android.com/studio/test) |
| Dart Testing | https://dart.dev/guides/testing |
| An introduction to integration testing | https://flutter.dev/docs/cookbook/testing/integration/introduction |
| An introduction to Unit Testing | https://flutter.dev/docs/cookbook/testing/unit/introduction |

# 2 System Architecture

## 2.1 Architectural Design

The Form Scriber mobile application is based on the flutter framework using Dart as the programming language. Flutter is an open-source user interface development kit created by Google. It is used to develop an application for Android, iOS, Linux, Mac, Windows, and the Web from a single codebase. Dart is a client-optimized programming language for apps on multiple platforms. Google also develops it to build mobile, desktop, server, and web applications. In Flutter, every component is called a widget. In each component, there exists a class. Dart has two types of widgets: stateless, which is used for static components, and stateful. Note that the static components are the ones that have no state change or nothing change within them as the user is exploring the application. The stateful widgets are the dynamic components. These components contain the state. The state is the information that can be read synchronously when the widget is built, and it might change during the lifetime of the widget.

The Form Scriber mobile application is developed using these two types of widgets. The application includes the following dart files:

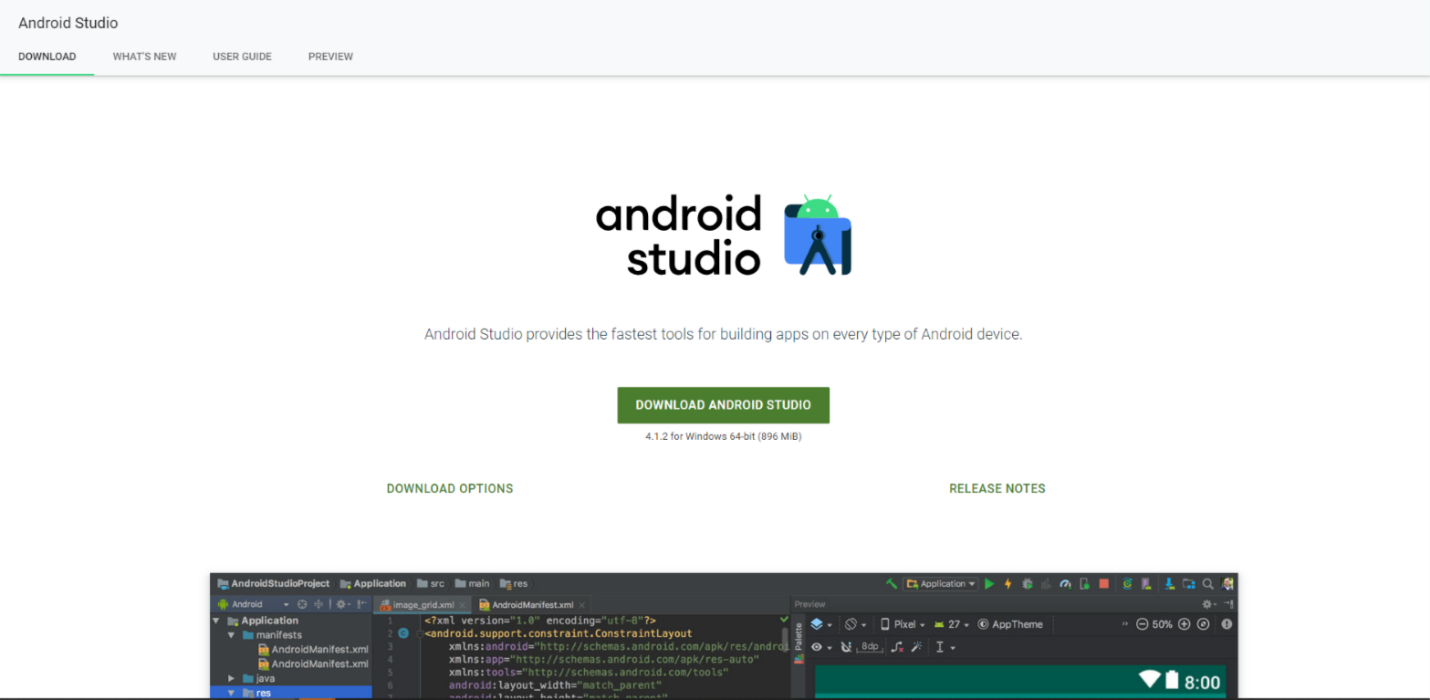
* main.dart file: This file is the driver of the entire application. It contains the main class from which the application is launched.
* Homepage. Dart: This file contains a stateless widget to display all the components, such as buttons and options available to navigate. There is a button for starting the conversation between a doctor and a patient. There is another button for system settings available for the user when trying to make mobile device changes.
* Help. Dart: Tis a stateless widget that contains some helpful information in case the user needs it while surfing through the application. This widget has some frequently asked questions and answers for the user to check on whenever they get lost, or the user does not know what to do with some particular actions.
* Settings. Dart: This is a stateful widget to maintain the state of the settings. This widget allows the user to change settings. It also stores the settings in persistent storage.
* Conversation. Dart: This is a stateful widget because its contents will be changed from time to time based on the content of the conversation between two people (doctor/physicians and patients)
* Routes. Dart: This is the routing file responsible for redirecting to the appropriate page once a link/button to a particular page is activated or clicked.
* App\_drawer. Dart: This is a stateless widget built for a sliding menu. The sliding menu will contain the following buttons: home, begin conversion, view reports, settings, help, and logout.

# 3 Setting up the application

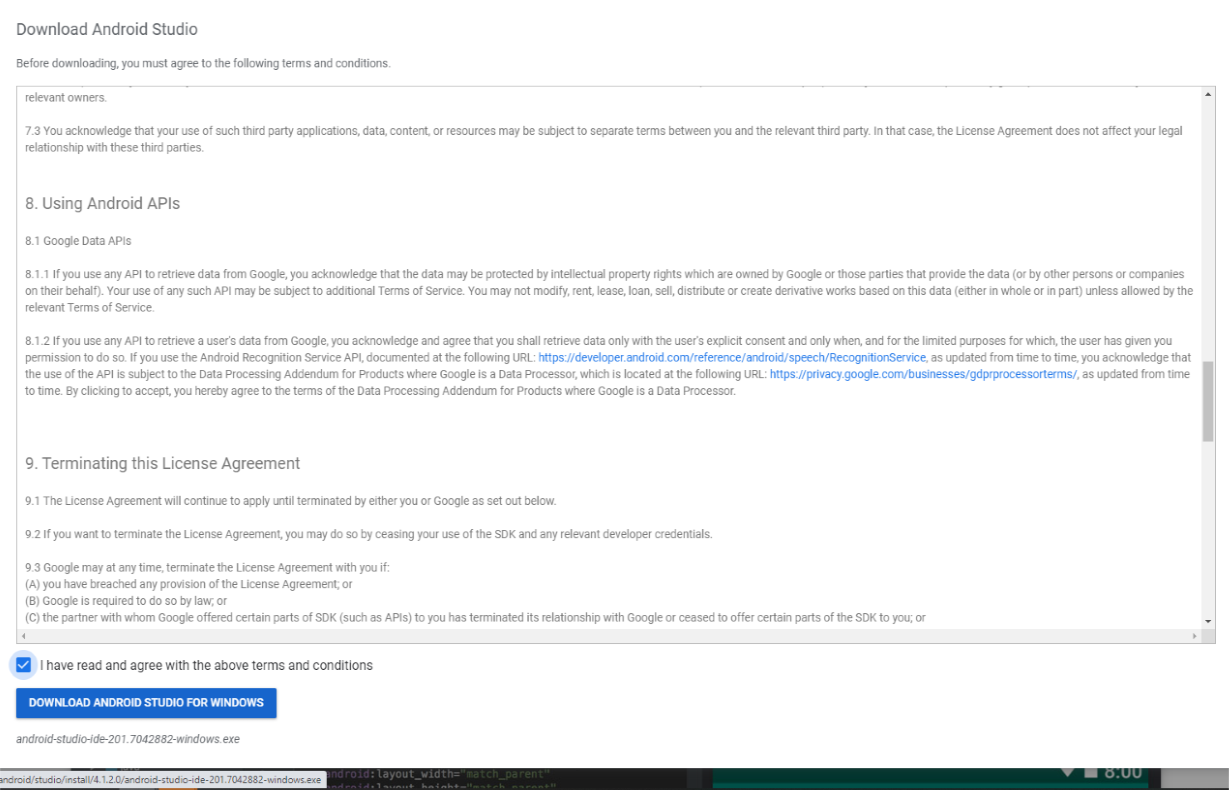
## 3.1 Downloading Android Studio

Reference <https://developer.android.com/studio/#downloads> for installation guidance for Windows, Mac, Linux, and Chrome OS.

1. Click the DOWNLOAD ANDROID STUDIO button.

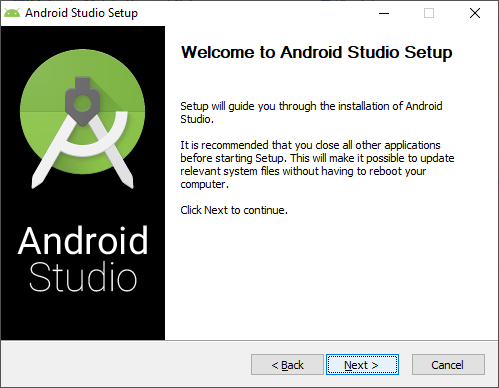


1. Click the checkbox “I have read and agree with the above terms and conditions,” then click on the DOWNLOAD ANDROID STUDIOS button.

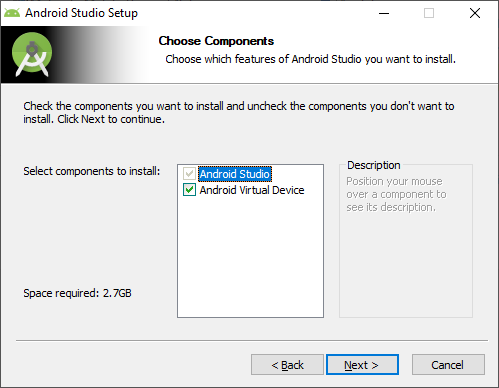


1. Click on the Save File button to download the file.

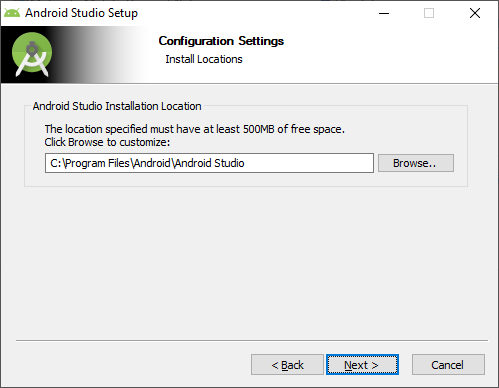
1. Once the download is completed, open the file to run it.



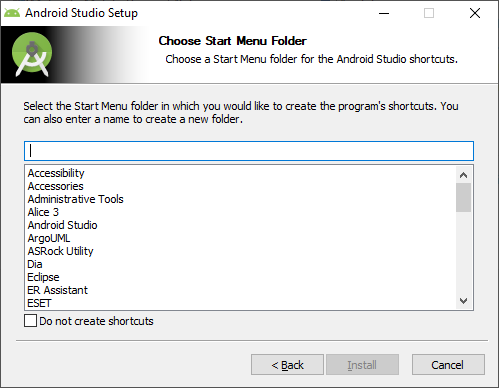
1. Click the Next button on the Welcome Screen.



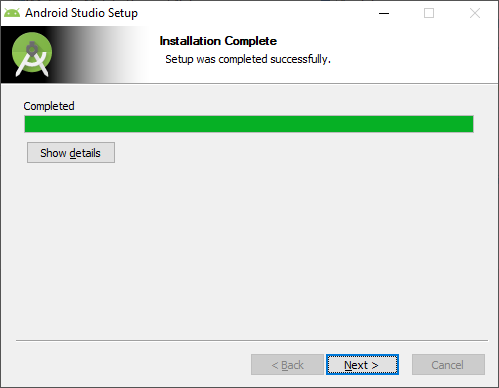
1. Click Next to install the components to continue.



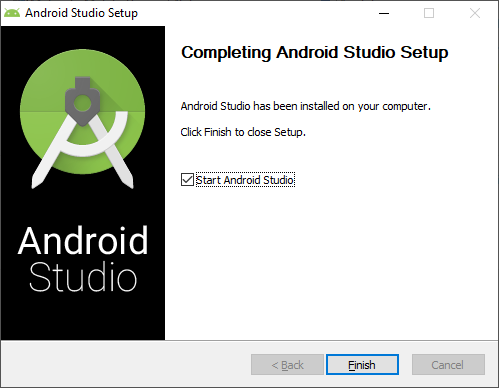
1. Choose a path for the installation in the following prompt and click Next.



1. Choose a Start Menu folder or Click the box for Do not create shortcuts. Click Install.

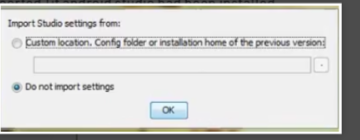


1. Once installed, a successful Window will appear. Click Next.



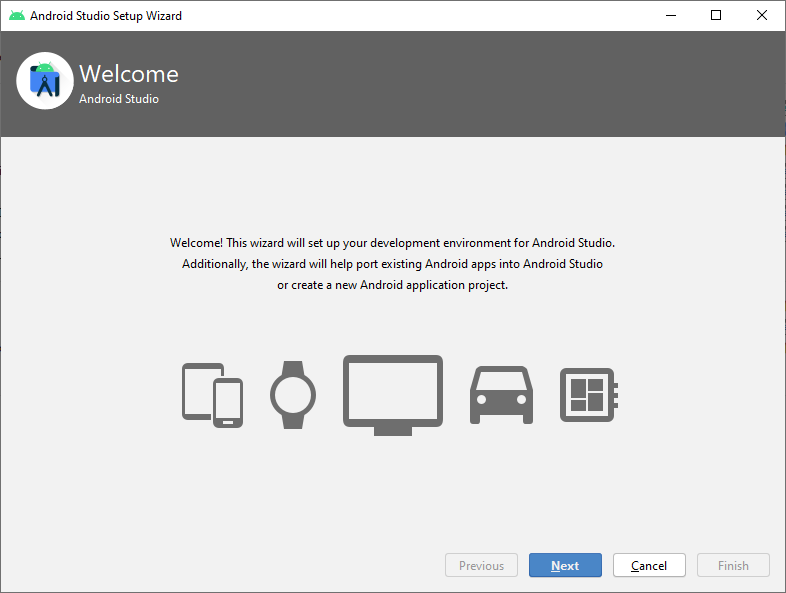
1. Completing Android Studio Setup will appear. Click Finish.

1. The next step will display a menu with radio buttons asking where to import the Studio settings from. Click the radio button for Do not import settings.

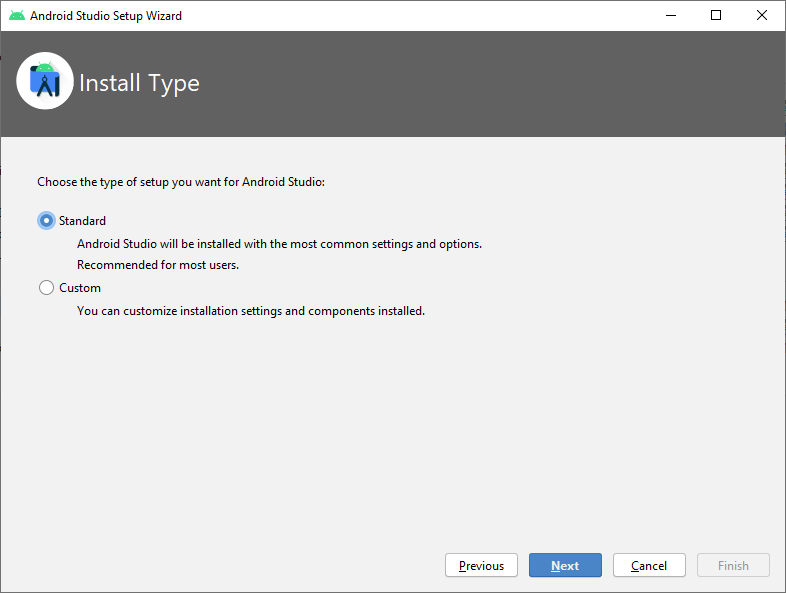


1. Android Studio will start and will automatically look for SDK components.

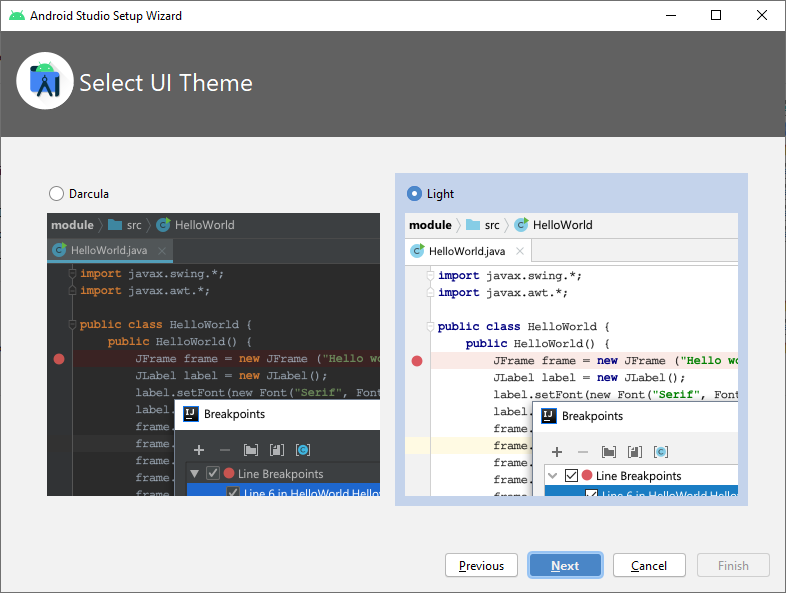
1. The Welcome box will appear.



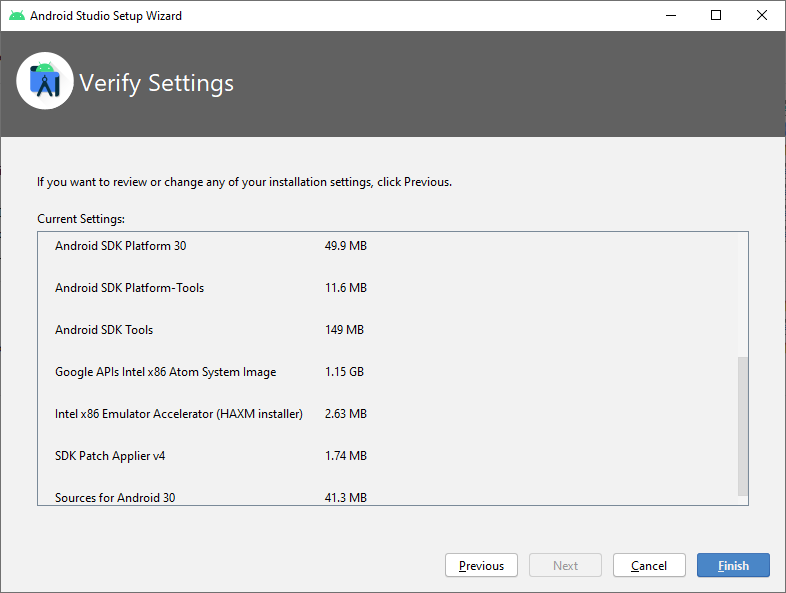
1. Click Next.



1. The Install Type box will appear for setup. Select the radio button for Standard, then click Next.



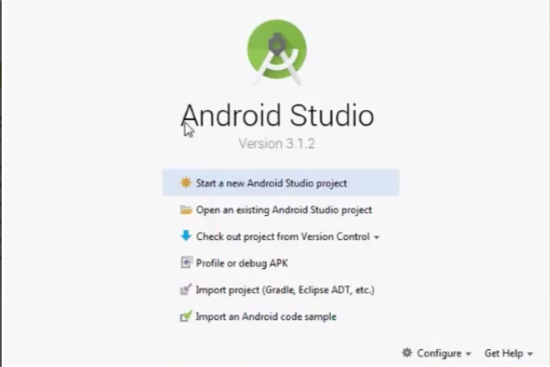
1. The next screen will ask you to Choose a Light theme or Dark theme (Dracula). Choose one, then click the Next button.



1. The next screen will ask if you want to review or change any of your installation

 Settings. Click Previous to change, or click Finish to download components.

18.   The Android Studio Welcome Page menu is loaded.



## 3.2 Downloading Flutter and Dart

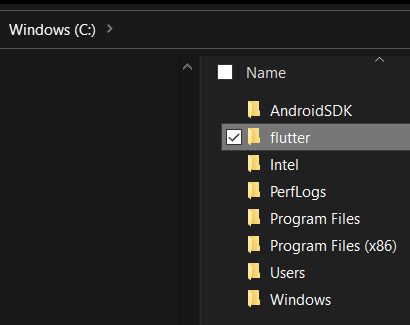
This section will explain the process of downloading Flutter and Dart.

Reference <https://flutter.dev/docs/get-started/install> for installation guidance for Windows, macOS, Linux, and Chrome OS.

Install Git. This is required for Flutter.

The aid below will step through installing Flutter on a Windows 10 machine.

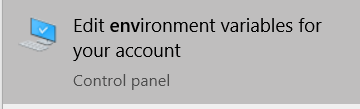
1. Download Flutter.
2. Unzip the contents of the zip file: 

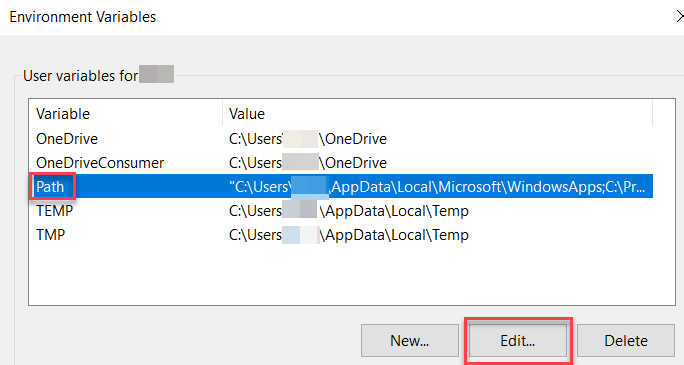
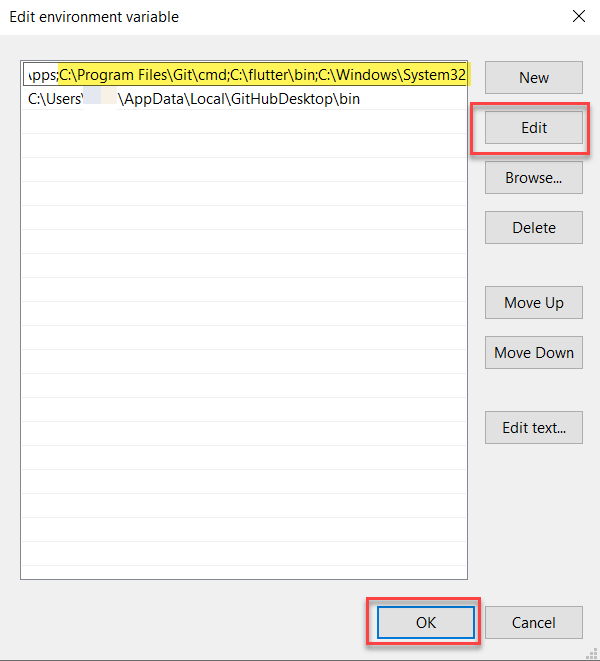
1. Copy the entire folder to your desired location. In this case, the C: drive: 

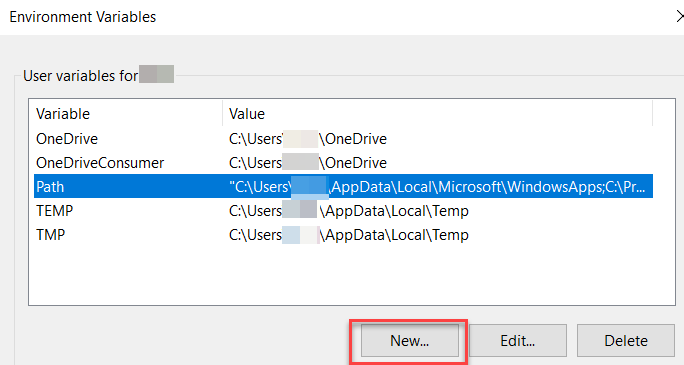
1. To run Flutter in PowerShell, the following variables must be added to your PATH environment variable:
2. Type **env** in your search bar:

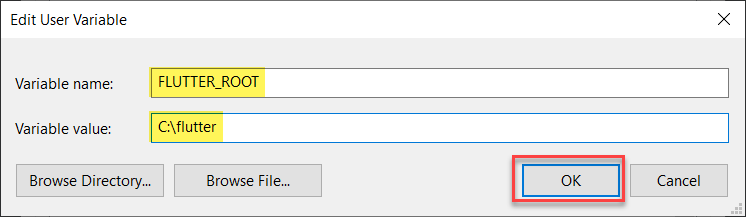


1. Select **Edit environment variables for your account**:



1. Select the **Path** variable. Click **Edit**to modify the variable.  
2. Add the values to your flutter/bin location following to **Path**:
3. C:\flutter\bin *(The location of your Flutter folder)*
4. C:\Program Files\Git\cmd *(The location of your Git install)*
5. C:\Windows\System32
6. 
7. Click **OK**on the **Edit environment variable**window.
8. Add the **FLUTTER\_ROOT**variable.
9. Click **New** on the **Environment Variables**window:

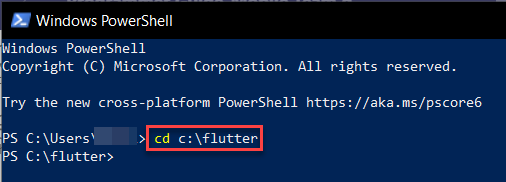
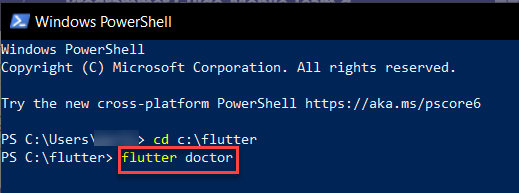
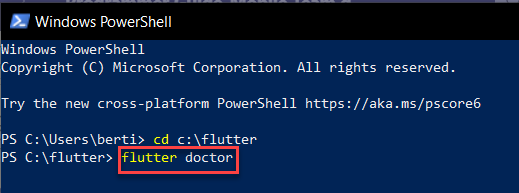


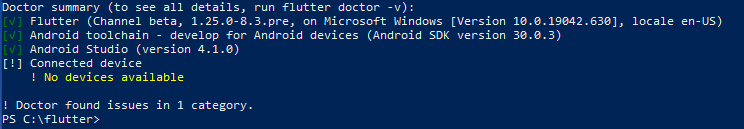
1. Enter the values below:
2. Variable Name: FLUTTER\_ROOT
3. Variable Value: *<the location of your Flutter folder>*
4. Click **OK.**

1. Click **OK** on the **Environment Variables**window.

1. Open **PowerShell**



1. Change to the location of the **flutter** folder copied in previous steps:  
2. Run the command **flutter doctor** to determine additional components needed for Flutter: 

1. If all components are installed successfully, the report should be similar to below: 

## 3.3 Frontend Development

### 3.3.1 Landing Page

The Landing Page is the first page that the user is greeted with when opening the application from an Android or iOS device. Users are shown the Form Scriber label to the top of the page, followed by instructions to login with their Google credentials. Two buttons are located in the middle of the page, the first button is the Login button, and the second is the Help button. The Login Button will redirect the user to the Longin screen, and the Help button will redirect the user to the Help screen. At the bottom of the screen, the user is given a “No Warranty” disclaimer.

### 3.3.2 Login page

The login screen will display a single “Google sign-in" button to sign users into the FormScriber application with their Google accounts. Clicking this sign-in button will display username/email and password input fields and a sign-in button to initiate the authentication process. Suppose the user’s credentials can’t be authenticated with Google. In that case, a message will be displayed on the Login page informing the user that the username and password they entered was invalid. If the user is authenticated with Google but cannot be authenticated with Firebase, a message will be displayed on the Login page informing the user that their credentials could not be authenticated with the selected Firebase database.

### 3.3.3 Home page

The Home page is the first page the user will see once they have been authenticated through the Login page.  This page displays navigation buttons to all application pages, thus acting as a “main menu.” The buttons displayed on the Home page will include the “FormScribe a Form” button, the “Settings” button, the “Logout” button, and the “Help” button.

### 3.3.4 Select Form Template page

The Select Form Template page is the first page of the “FormScribe” process and is accessed when the user clicks the “FormScribe a Form” button on the Home page. At the top of the page, the sentence “Step 1 of 2: Select a form template” will be displayed to let the user know where they are in the FormScribe process. A dropdown button on this page will be populated with the form templates available in the user’s “FormScriber form templates” directory of Google Drive. This dropdown allows the user to choose which form template they wish to FormScribe into a report. If no form templates are available in the user’s “FormScriber form templates” directory, then a dialog window with an “Ok” button will immediately open. This dialog window will inform the user they must create a form template and store it in their “FormScriber form templates” directory of their Google Drive account before they FormScribe a form. When the user clicks the ‘X’ button or the “Ok” button, the user should be returned to the Home page. If form templates do exist, they will be populated in the dropdown button. The user must then select the template they wish to use in the “FormScribe” process and type in a name for the FormScribed report into a name input textbox. Once these two fields are completed, the user can click the “Continue to Step 2: Record form entries” button to attempt to move to the Record Form Entries page. If both fields are not filled, the “Continue to Step 2: Record form entries” button will be disabled to prevent the creation of null pointers. Before the user can move to the next step, the provided name of the FormScribed report must pass a uniqueness test with the existing FormScribed reports in the “FormScribed Reports” directory of the user’s Google drive. If the name is unique, the Record Form Entries page is opened to continue with the FromScribe process. If the name already exists, an error message stating “A FormScribed report with that name already exists. Please choose a different name.” will be displayed below the FormScribed report name input field. Once the user has changed the name, they may reattempt to continue through the process by pressing the “Continue to Step 2: Record form entries” button again.

### 3.3.5 Record Form Fields page

The Record Form Fields page is the second and final page of the “FormScribe” process and is accessed from the Select Form Template page. At the top of the page, the sentence “Step 2 of 2: Record form fields” will be displayed to let the user know where they are in the FormScribe process. This page contains a display area, an input textbox, a recording button, and a “Complete my FormScriber report” button.  The display area is used to display responses returned from DialogFlow and the input provided by the user. The input textbox allows the user to type in the form field names and values to be identified and recorded into the “FormScriber report.”

In contrast, the recording button allows the user to speak the field names and values for the same purpose. The “Complete my FormScribed report” button is intended to initiate the final steps of the “FormScribe” process. The first step will be displaying a dialog window with an “Ok” button that informs the user about the status of their “FormScriber report.” Suppose the “FormScriber report” is still accessible. In that case, this dialog will inform the user that their “FormScriber report”: with the name they specified has been completed and can be found in their "FormScriber reports" directory on Google Drive. If the “FormScriber report” is no longer accessible at this point, or an error (such as connection failure) happened anywhere during the “FormScribe” process, the dialog window will inform the user of this issue in completing the FormScribe process. In both successful and failed FormScribe processes, when the user closes this dialog window, using either the 'X’ button or the "Ok" button, the user will be returned to the homepage of the FormScriber app. Suppose, at any time in the FormScribe process, the user attempts to leave the page they are on. In that case, they will be provided with a confirmation dialog asking if they are sure they wish to leave the page and notifying them that leaving will cause all “FormScriber report” processes to be deleted. If the user clicks yes in this dialog to leave the current page, the “FormScriber report” that may have been created in this process will be deleted, and the user will be redirected to the page they chose.

### 3.3.6 Settings page

The settings page has a checkbox to enable/disable dark mode, a font family selector, a font size selector, and an apply button. The settings are saved to persistent storage via the shared\_preferences dependency in pubspec.yaml once the user taps the apply button. The settings are read from the persistent storage and stored in a class called “Settings” when the application launches.

## 3.4 Backend Development

The backend of the FormScriber application leverages a variety of existing software solutions to simplify its design. Each of these solutions were chosen for specific reasons, described later in this section.

The following software solutions were used in this application:

* The Flutter framework and Dart programming language were used to create a cross-platform application.
* A Google Firebase Realtime database and Google accounts were used for an all- in-one management system for managing users, access control, and application information.
* Google Drive was used as the central repository for form templates and FormScriber report storage.
* A variety of Dart library packages were imported to handle much of the communication with the solutions above, and a few were used to for handling internal business logic.

The decision to use Flutter as the UI framework was made according to two factors:  the development team’s desire to distribute the application to as many platforms as possible and their predominant coding experience in Java. Flutter’s ability to create cross-platform applications and Dart’s syntax similarities to Java made it the appropriate choice for this application. Additionally, Flutter offered many well-maintained, first-party Dart library packages from the Pub package manager that we could incorporate to simplify our designs. The Dart library packages used in this application were the “dialog\_flowtter,” “firebase\_auth,” “firebase\_core,” “get\_it,” “googleapis,” “googleapis\_auth,” and “google\_sign\_in” packages, all of which were imported into the application using Flutter’s “Pub” package manager. The usage of each of these packages will be described in further detail later. The design of the application was centered around the use of Google Drive.

Google Drive documents are being used as the basis for the application’s “FormScribe” process. Google Drive eliminates the need for a custom document management server. Because of this, Google Drive was selected to become the foundation of the application’s architecture. A user sign-in with Google credentials is required for authentication since Google Drive requires users to have a Google account.

The sign-in implementation was simplified with the  “google\_sign\_in” package, which provided the API for authenticating a Google user’s credentials and returned an Open ID Connect token, a short-lived access token, and an authorization code. This access token is used to maintain authorization to use Google APIs. The authorization code is used to refresh the access token when it expires. The open ID token is not used in this application.

To access Google Drive functions in Flutter, the “googleapis” package is used with the user’s access token. If the user’s token expires, the “googleapis\_auth” package and the authorization code are used to refresh the token. Google Drive documents involved with this application are considered either a “form template” or a “FormScriber report.” An unfilled document that the user wishes to fill out through the “FormScribe” process is considered a “form template,” and a document that has been filled out through the “FormScribe” process is considered a “FormScriber report.”  The input selection of a “form template” from and output of a “FormScriber report” to Google Drive requires using the Drive API found in the “googleapis” package and directories specific to “form templates” and “FormScriber reports,” respectively. Storing these directory paths in the application itself is insecure. If these paths were updated in the future, it would be challenging to update all mobile devices. Additionally, the authentication system lacked a way to track real-time changes in user’s role-based access. The simplest solution seemed to be Google Firebase.

Since the application used Google DialogFlow within a Google Cloud Project, Firebase was part of the Google Cloud. Firebase provided many administrative functions for managing user’s roles, access control, user logins, and application data. Selecting Firebase as a part of our management system was simple. Additionally, the development team desired the management system to be easily understood by a broader population and would be easy to integrate into our design plans; therefore, given the widespread use and extensive documentation of Google products and services, Firebase was the optimal choice.

Firebase offers two database options to users: the Realtime Database and the Cloud Firestore. The Realtime Database was chosen over the Cloud FireStore Database for several reasons. First, the application’s stored data would be simple enough to be stored in the Realtime Database’s JSON tree. Secondly, the financial cost for database operations would likely be less using the bandwidth-and-storage-based cost of the Realtime Database versus the operations-performed-based cost of the Cloud FireStore Database. Finally, the lower latency of the Realtime Database was preferable for regular state syncing of logged-in users. The packages “firebase\_auth” and “firebase\_core” provided the APIs to manage access to a firebase instance, manage login of the current user, and fetch the application's configuration settings securely. To make the firebase instance available throughout the application, the package “Get\_it” is used to create a resource locator as a global variable. This resource locator references the firebase instance, thus making the instance available throughout the app.

## 3.5 Running Tests

Three testing tools can be used for testing the Mobile Application.

* Android studio
* Flutter test package.
* DevTools

### 3.5.1 Android studio

Android Studio offers an easy way for testing through JUnit, which runs on the local JVM, to include running tests on mobile devices (Studio, 2021). Testing can also be integrated with the following frameworks:

* Mockito – for testing Android API calls in unit tests (Studio, 2021).
* Espresso – can be done quickly by using Espresso Test Recorder (Studio, 2021).
* UI Automator – used for user interaction and can be done with Espresso Test Recorder (Studio, 2021).

### 3.5.2 Test types and location

Based on the test that will be run, Android Studio has source code directories called source sets for Local Unit Test and Instrumented Tests (Studio, 2021).

Instrumented Tests are located in the following directory **module-name**/src/androidTest/java/. When running tests to access the context of the Form Scriber application, on an emulator, or directly through a live mobile device, the instrumented Tests are used (Studio, 2021). Make sure when running this test to give them their own AndroidManifest.xml file.

Creating a local unit test or instrumented test, follow these steps for testing a class or method:

1. Open the Java file containing the code you want to test.
2. Click the class or method you want to test, then press Ctrl+Shift+T.
3. In the menu that appears, click **Create New Test**.
4. In the **Create Test** dialog, edit any fields and select any methods to generate, and then click **OK**.
5. In the **Choose Destination Directory** dialog, click the source set corresponding to the type of test you want to create: **androidTest** for an instrumented test or **test** for a local unit test. Then click **OK**.

Specify the test library dependencies in the app module's build.gradle file:

dependencies {  
    // Required for local unit tests (JUnit 4 framework)  
    testImplementation 'junit:junit:4.12'  
  
    // Required for instrumented tests  
    androidTestImplementation 'com.android.support:support-annotations:24.0.0'  
    androidTestImplementation 'com.android.support.test:runner:0.5'  
}

To run a test, proceed as follows:

1. Be sure your project is synchronized with Gradle by clicking **Sync Project** https://developer.android.com/studio/images/buttons/toolbar-sync-gradle.png in the toolbar.
2. Run your test in one of the following ways:

* In the **Project** window, right-click a test and click **Run** https://developer.android.com/studio/images/buttons/toolbar-run.png.
* In the Code Editor, right-click a class or method in the test file and click **Run** https://developer.android.com/studio/images/buttons/toolbar-run.png to test all methods in the class.
* To run all tests, right-click on the test directory and click **Run tests** https://developer.android.com/studio/images/buttons/toolbar-run.png.

### 3.5.3 Flutter Test Package

Running the Flutter test package for testing widgets.

To run a unit test, the following steps must be obtained (Flutter, 2020).

1. Add the test or flutter\_test dependency.
2. Create a test file.
3. Create a class to test.
4. Write a test for our class.
5. Combine multiple tests in a group.
6. Run the tests.

Test package, writing tests in Dart must include the following (Flutter, 2020):

Dev\_dependencies:

Test: <latest\_version>

**Creates a test file**

Create two files: counter.dart and counter\_test.dart (Flutter, 2020). The counter.dart file contains a class to test, and resides in the lib folder (Flutter, 2020). The counter\_test.dart file contains the tests themselves and lives inside the test folder (Flutter, 2020).

counter\_app/

lib/

counter.dart

test/

counter\_test.dart

## 3.6 Running end-to-end tests

End-to-end testing along with integration testing is done to monitor the Form Scriber application. Integration tests have been implemented for emulators of Android and iOS mobile devices.

The flutter driver package is used to set up an instrumented application and lead those application tests from a test suite (Dart, 2021).

The following method is used to test the Form Scriber application:

1. Create a Test
2. Add the flutter\_driver dependency.
3. Create the test files.
4. Instrument the app.
5. Write the integration tests.
6. Run the integration test.

**Create an App to Test**

Create a .dart class for testing widgets in the Form Scriber application.

**Add the Flutter Driver Dependency**

The flutter\_driver package will be used to write integration tests. Add the flutter\_driver dependency to the dev\_dependencies section of the app’s pubspec.yaml file (Flutter, 2020).

Also, add the following test dependency for test functions and assertions (Flutter, 2020)

dev\_dependencies:

flutter\_driver:

sdk: flutter

test: any

**Create Test Files**

Two files will be created in the same directory and named test\_driver (Flutter, 2020).

1. The first file will have an “instrumented” version of the app. This test will record performance profiles from a test suite (Flutter, 2020). The file created is called driver/app. Dart
2. A second file will have the test suite, which will verify it runs as it should and record performance profiles. The second file created is called test\_driver/app\_test. Dart

This creates the following directory structure:

counter\_app/

lib/

main.dart

test\_driver/

app.dart

app\_test.dart

**Instrument the App**

Instrument the app with the following two steps:

1. Enable the flutter driver extensions.
2. Run the app.

The following code will go in the test\_driver/app.dart file (Flutter, 2020).

import 'package:flutter\_driver/driver\_extension.dart';

import 'package:counter\_app/main.dart' as app;

void main() {

// This line enables the extension.

enableFlutterDriverExtension();

// Call the `main()` function of the app, or call `runApp` with

// any widget you are interested in testing.

app.main();

}

Then create the following for testing:

1. Create [SerializableFinders](https://api.flutter.dev/flutter/flutter_driver/CommonFinders-class.html) to locate specific widgets.
2. Connect to the app before our tests run in the desired function.
3. Test the essential scenarios.
4. Disconnect from the app after the tests complete.

When ready, test the application through the Android Emulator and then the Ios Simulator, or connect a mobile device for testing.

Run the following command from the root of the project (Flutter, 2020):

flutter drive --target=test\_driver/app.dart

The command will do the following:

* --target, builds the test and installs it on the emulator or real-time device.
* Allows the app to be launched.
* Runs the test suide located in the test\_driver/ folder, app\_test.dart

# 4 Code structure

The application operates with a framework of segregated user interface code and business logic code. Flutter is used as a framework, with the Dart programming language, and importing various packages and libraries from the open-source package repository <https://pub.dev/>.

In Flutter, each screen presented on the mobile display is referred to as a Route. Each Route can consist of many elements drawn on screen, called Widgets. Widgets can be Stateless or Stateful. A stateless widget is a widget that describes part of the user interface by building a collection of other widgets that describe the user interface more concretely. Stateful widgets, on the other hand, will typically rebuild many times during the application's lifetime using a State.setState setter, or depending on InheritedWidgets.

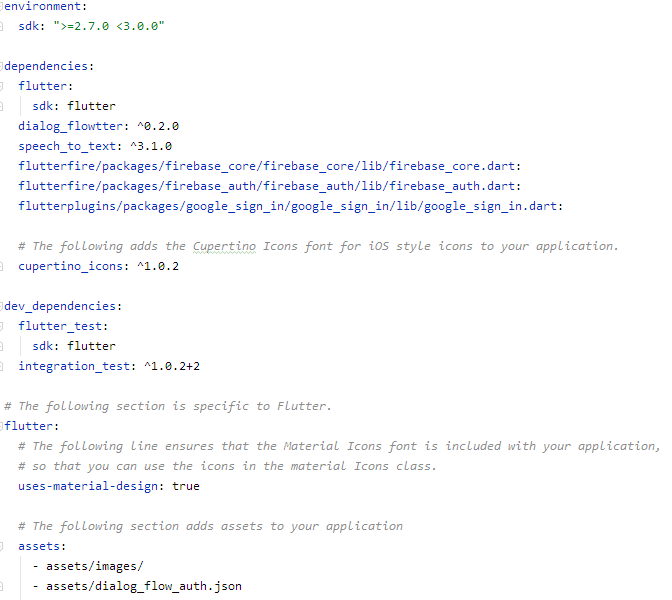
## 4.1 Frontend

The application operates with a framework of segregated user interface code and business logic code. Flutter is used as a framework, with the Dart programming language, and importing various packages and libraries from the open-source package repository <https://pub.dev/>.

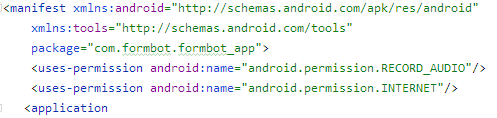
In Flutter, each screen presented on the mobile display is referred to as a Route. Each Route can consist of many elements drawn on screen, called Widgets. Widgets can be Stateless or Stateful. A stateless widget is a widget that describes part of the user interface by building a collection of other widgets that describe the user interface more concretely. Stateful widgets, on the other hand, will typically rebuild many times during the application's lifetime using a State.setState setter, or depending on InheritedWidgets.

4.1.1 Dependencies

The following dependencies are required for application functionality. Packages will be imported at the class level in the respective dependent Dart files, and not all package versions are compatible with the recent versions of Dart. For example, speech\_to\_text is dependent on Dart SDK version 2.7.0, so the minimum SDK version is set to 2.7.0. The pubspec.yaml file defines this Flutter project’s package dependencies and is used for resolving dependency trees during pub get



In addition to the package dependencies, the application needs to be granted permission to record audio and connect to the internet. For Android, these are defined in the AndroidManifest.xml located in: formbot\_app/android/app/src/main/AndroidManifest.xml

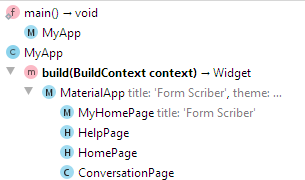


To overcome phone compatibility issues related to speech\_to\_text, the minimum Android SDK version must be set in the build.gradle file of the Android app directory: formbot\_app/android/app/build.gradle

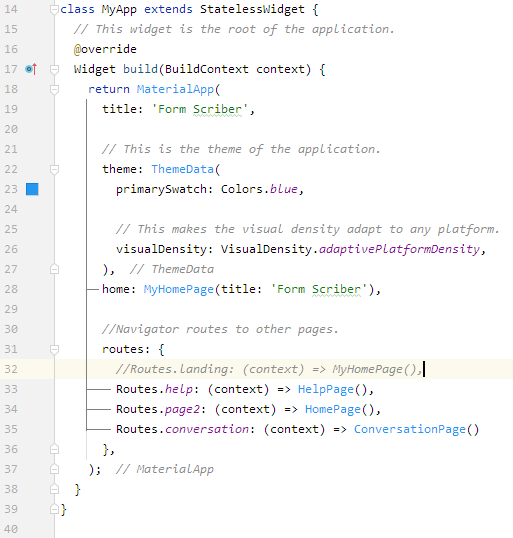


### 4.1.2 main.dart

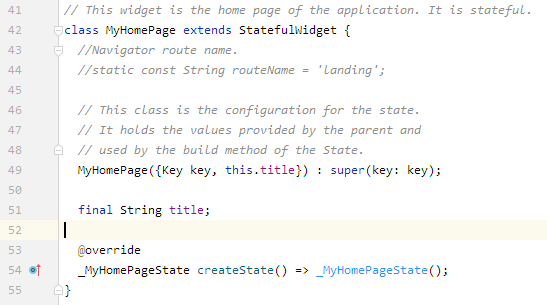
Main is the application starting point containing the function main(), where Flutter will begin code execution. Below is an outline of the user interface classes within Main:



The void function main() calls the function runApp to Inflate the MyApp widget and attach it to the screen. MyApp is the StatelessWidget holding the root of the application and Navigation Routes for other pages starting at MyHomePage.



Following the initialization of MyApp the MyHomePage screen is built based on the state \_MyHomePageState.

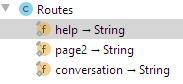


State classes act as logical controllers of their respective StatefulWidget class. The \_MyHomePageState State includes a Login button and a Help button.

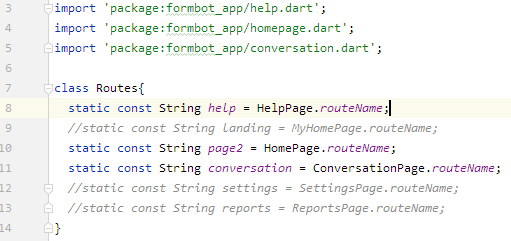


4.1.3 Routes/routes. Dart

The Routes class defines the strings which point the Navigator to different pages.



There are placeholders currently commented out.



Each class imports the routes.dart class, and navigation can occur as in this example:

*// Login Button*   
*RaisedButton*(   
  onPressed: () => {Navigator.*pushNamed*(context, Routes.*page2*)},)

### 4.1.4 HomePage

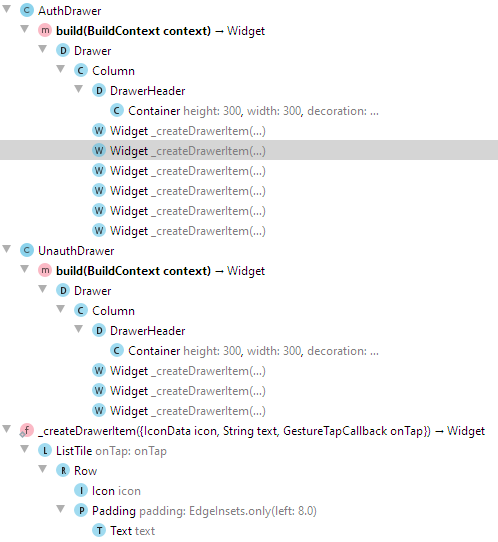
The HomePage is a StatelessWidget as it does not have any dynamic elements within its body. It does however include an asynchronous Future boolean method \_willPopCallback to allow coming back to the home page without rebuilding after navigating to another Route.





### 4.1.5 App\_drawer. Dart

This class holds the sliding menu and creates both menus for authorized users and unauthorized users.



To make this work, each screen/page/file would need to include the following:

import 'widgets/app\_drawer.dart';

or

import 'package:formbot\_app/widgets/app\_drawer.dart';

A routename value will need to be added as well:

static const String *routeName*= 'help';

whereas the routename is the name of the page.

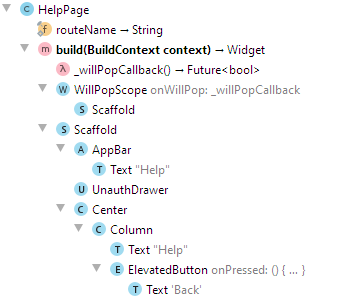
class HelpPage extends StatelessWidget {   
  static const String *routeName*= 'help';

Then the menu/drawer would need to be added to each screen as follows:

// Select the drawer that is needed   
//  AuthDrawer - Authorized Users   
//  UnauthDrawer - Unauthorized Users   
endDrawer: AuthDrawer(),

### 4.1.6 Help. Dart

The Help page is a StatelessWidget without any dynamic widgets within its body. It includes an asynchronous Future boolean method \_willPopCallback to allow backward navigation.

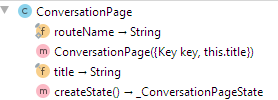




### 4.1.7 Conversation. Dart

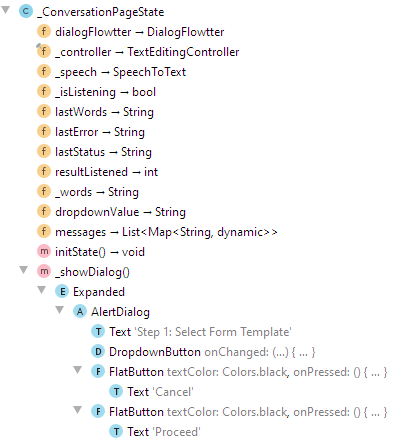
The conversation screen is a StatefulWidget which contains the framework for all of the following:

1. Establishing the connection to DialogFlow,
2. Selecting a Form Scriber template doc based on the established Google Drive repository.
3. Sending and receiving messages to and from DialogFlow through text or voice.



The  \_ConversationPageState builds the context for the ConversationPage widget, and initializes the state of 3 components:

1. DialogFlowtter instance
2. SpeechToText instance
3. \_showDialog, an AlertDialog which prompts the user to select the Google Drive doc URL before starting speech.





# Appendices:

# Appendix A – References

Dart. (2021, February 1). *Dart Testing*. Retrieved from Dart: https://dart.dev/guides/testing

Flutter. (2020, December 9). *An introduction to integration testing*. Retrieved from Flutter: https://flutter.dev/docs/cookbook/testing/integration/introduction

Flutter. (2020, September 11). *An introduction to Unit Testing*. Retrieved from Flutter: https://flutter.dev/docs/cookbook/testing/unit/introduction

Studio, A. (2021, February 24). *Test Your App*. Retrieved from Google Developers: https://developer.android.com/studio/test