Programmer Guide

TEAM: CHARLIE

Presented by: Michael Le, Debashis Jena, Austin Johnson, Prince Antwi Aboagye, Didimus Kimbi, Damion Sevilla

SWEN 670 – SOFTWARE ENGINEERING PROJECT

JULY 23, 2021

REVISION 2.0

Project name: Mnemosyne, Disability Mobile Application

Date: July 23, 2021

Project Leader: Michael Le

Phase: Design & Engineering and Execution

For approval: Michael Le

Michael le Date: 07/23/2021

For approval: Dr. Mir Mohammed Assadullah

_____ Date: 07/23/2021

Revision History

Version Number	Date	Description	Approved By
1.0	07/23/2021	Initial Programmer Guide Release	Michael Le
2.0	08/06/2021	Programmer Guide Updated	Debashis Jena

Table of Contents

1 Introduction

1.1 Purpose

The purpose of this Programmer's Guide documentation is to illustrate and demonstrate the technical software perspective of the Mnemosyne application. The futures development might base on this guidance to integrate more techniques into the tool used, the language selected, the architecture, and the operation of the application.

1.2 Intended Audience

Software Developers and Tester is the primary audience for current and future software development. The document provides specific details on technical programming guidance, assuming that it helps future software developers integrate more development without knowing the Mnemosyne application.

1.3 Technical Project Stakeholders

The table below shows a list of stakeholders involves in the project for the Mnemosyne Mobile Application:

Table 1 - Project Stakeholders

Name	Role
Dr. Mir Assadullah	Stakeholder (Project Owner)
UMGC Professors	Software Engineer Staff Department
Michael Le	Project Manager

Debashis Jena	Lead Developer
Austin Johnson	Developer
Didimus Kimbi	Developer
Damon Sevilla	Tester / CO-PM
Prince Aboagye	Business Analyst

1.4 References

Table 2 - Referenced Documents

Title	Reference
Install Flutter	http://flutter.dev
Android Studio additionally requires that a Java Development Kit (JDK) install	https://www.oracle.com/java/technologies/javase-downloads.html
Download Android Studio	https://developer.android.com/studio
Cloud Speech-to-Text setup	https://cloud.google.com/speech-to-text https://cloud.google.com/speech-to-text/docs/reference/rest/?apix=true

1.5 Definitions, Acronyms, and Abbreviations

Table 3 - Definitions, Acronyms, and Abbreviations

Acronyms and Abbreviations	Definitions
AVD	Android Virtual Device
APK	Android Application Package
AMD	Advanced Micro Devices
iOS	iPhone operating system
Flutter CLI	Flutter Command Line Tool
UI / API	User Interface / Application Program Interface
IDE	Integrated Development Environment
SDK	Software Development Kit
ENV	Environment Variables
JDK	Java Development Kit

2 System Architecture

2.1 Architectural Design

The Mnemosyne 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 Android, iOS, Linux, Mac, Windows, and Web applications 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 uses for static components, and stateful. Note that the static components are the ones that have no state change or nothing changes 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 reads synchronously when the widget builds, and it might change during the widget's lifetime.

The Mnemosyne mobile application development using these two types of widgets. The application includes the following dart files:

- Main.dartfile: This file is the driver of the entire application. It contains the main class from which the application launch.
- encryption_service.dart: This file contains an encrypted service package to encrypt and decrypt all the components of the generated text. Saving and viewing locally stored data is encrypted, the encryption algorithm service in this file.

- schedule_delete_text.dart: This file contains the flutter workmanager service components that manage the retention policy functionality, retaining or deleting data outside the seven-day retention policy.
- text_to_speech.dart: This is service provides flutter-based text to speech from the generated text (notes).
- audio_recognize.dart: This is a stateful widget; its contents will be changed from time to time while depending on the state of audio input and recognition.
- audio_recorder.dart: This is a stateful widget, functionality for recording and playing back wav file to the user of their voice for initial testing.
- view_notes.dart: This is a stateless widget; it loads note content from the menu.
- view_notes_details.dart: This is a stateful widget, content text map format managements.
- menudrawer.dart: This is a stateless widget, displays to screen highlighted and styled text, recognized speech.
- edit.dart: This is a stateless widget, contains a constructor to edit text.
- save.dart: This is a stateless widget, contains functionality for saving new notes.
- saveform.dart: This is a stateful widget, allows for the editing and saving of generated notes.
- script.dart: This is a stateless widget, bottom menu buttons for overall navigation within the application.
- recognized_content: This is a stateless widget, content changes from time to time-based on audio recognition.
- settings.dart: This is a stateless widget, menu functionality for accessing the settings page.

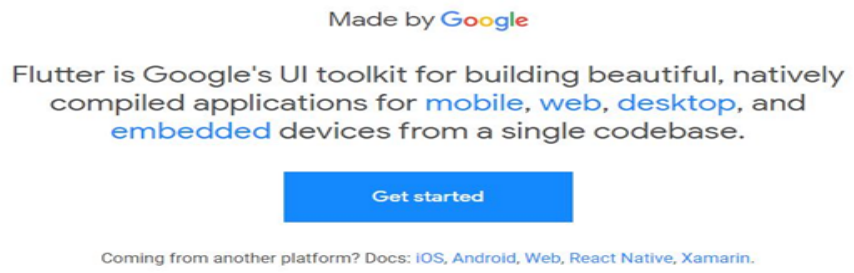
- textmap.dart: This file contains the text map for the JSON file. It is writes, reads, and converts content to the local document directory.
- local_auth_api.dart: Biometrics and local authentication services.

3. Setting up the Application

3.1 Installing Flutter and Dart

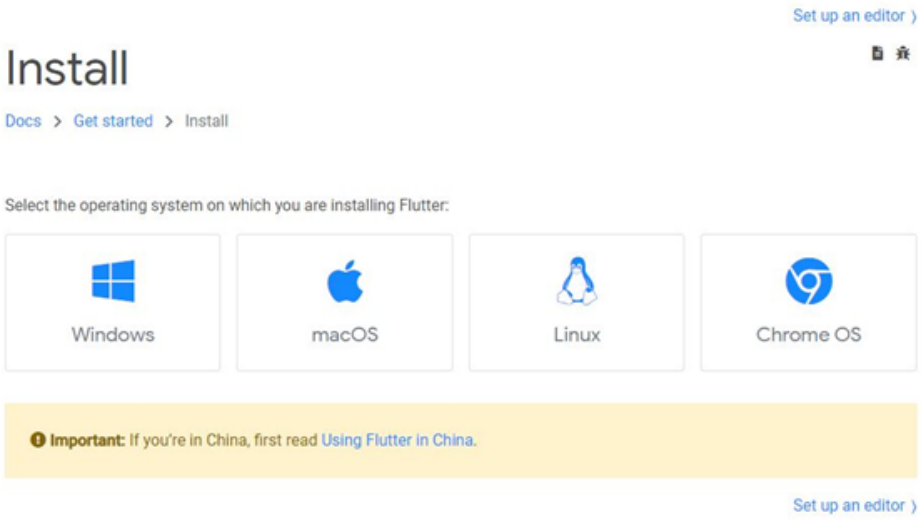
The following instructions assume a Windows environment, but the steps are largely the same for other supported operating systems.

1. Navigate to http://flutter.dev
2. Click the "Get started" button.

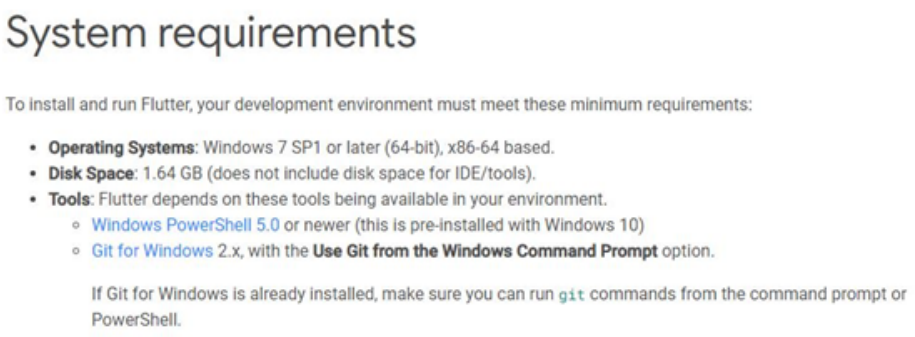


3. Select your operating system.

   In our example, we will use Windows.



4. If you do not already have them installed, use the links on the next page to install PowerShell and Git for Windows.



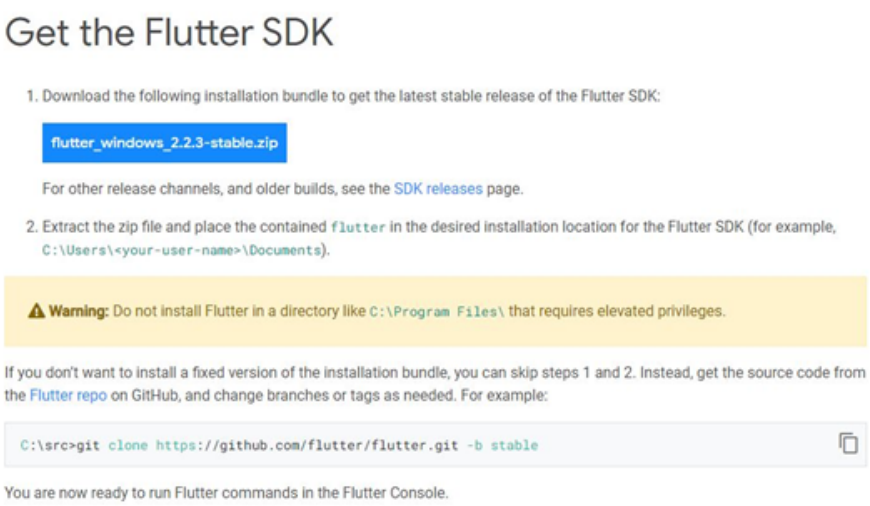
System requirements

To install and run Flutter, your development environment must meet these minimum requirements:

- **Operating Systems**: Windows 7 SP1 or later (64-bit), x86-64 based.
- **Disk Space**: 1.64 GB (does not include disk space for IDE/tools).
- **Tools**: Flutter depends on these tools being available in your environment.
  - Windows PowerShell 5.0 or newer (this is pre-installed with Windows 10)
  - Git for Windows 2.x, with the **Use Git from the Windows Command Prompt** option.

    If Git for Windows is already installed, make sure you can run `git` commands from the command prompt or PowerShell.

5. Download the Flutter SDK using the button below.



Get the Flutter SDK

1. Download the following installation bundle to get the latest stable release of the Flutter SDK:

   flutter_windows_2.2.3-stable.zip

   For other release channels, and older builds, see the SDK releases page.

2. Extract the zip file and place the contained `flutter` in the desired installation location for the Flutter SDK (for example, `C:\Users\<your-user-name>\Documents`).

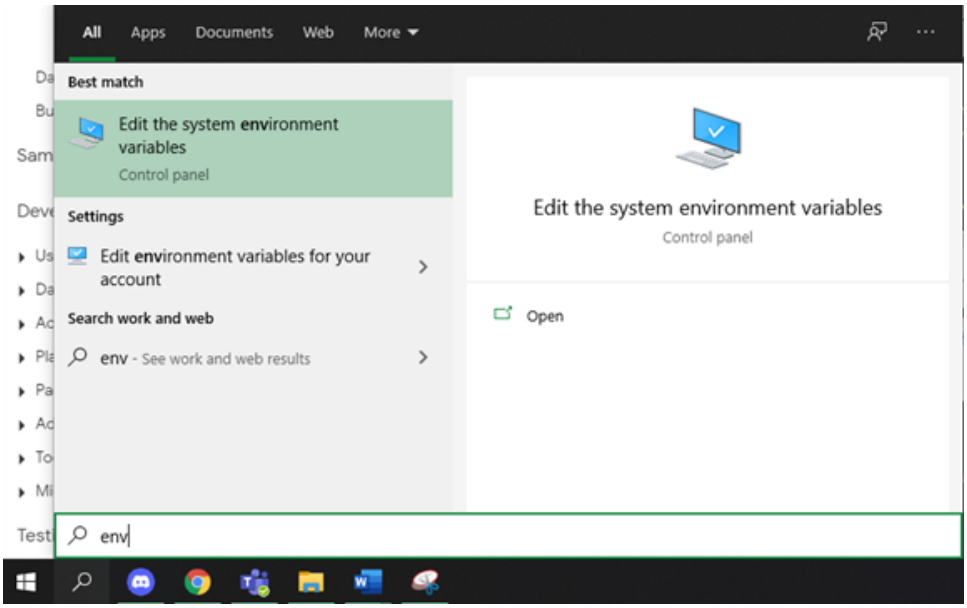
⚠ **Warning:** Do not install Flutter in a directory like `C:\Program Files\` that requires elevated privileges.

If you don't want to install a fixed version of the installation bundle, you can skip steps 1 and 2. Instead, get the source code from the Flutter repo on GitHub, and change branches or tags as needed. For example:

```
C:\src>git clone https://github.com/flutter/flutter.git -b stable
```

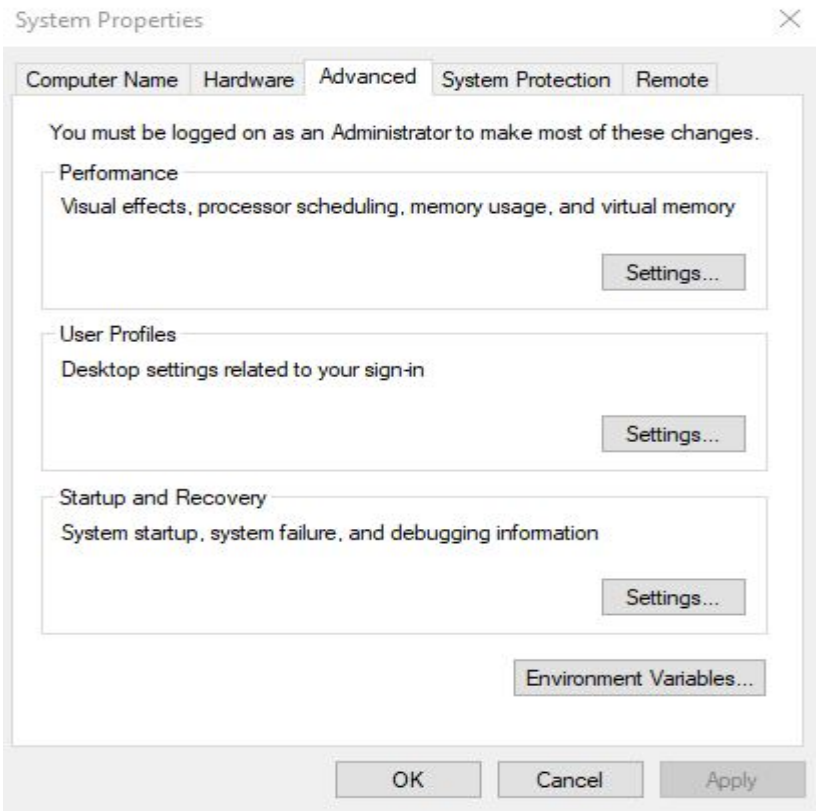
You are now ready to run Flutter commands in the Flutter Console.

6. After saving the zip file, extract the SDK to your desired location.
7. To enable flutter commands in the regular Windows command prompt, begin by clicking the search button and searching for "env."

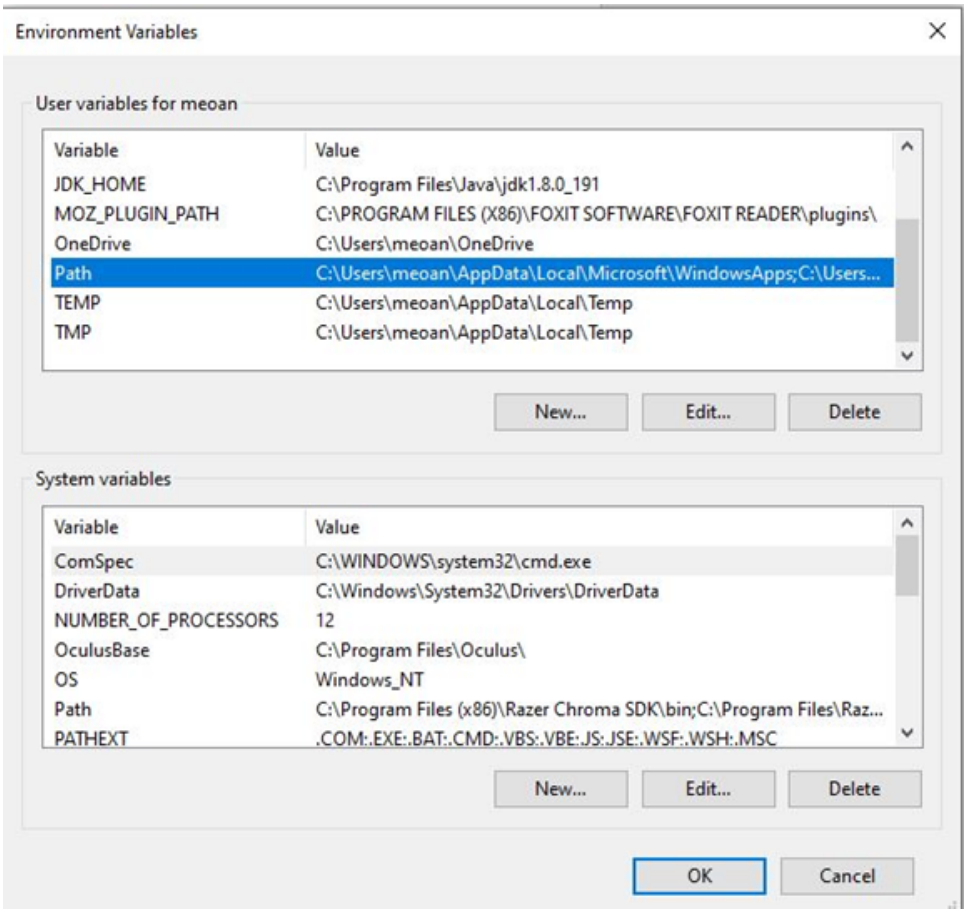


8. Click on Edit the system environment variables.

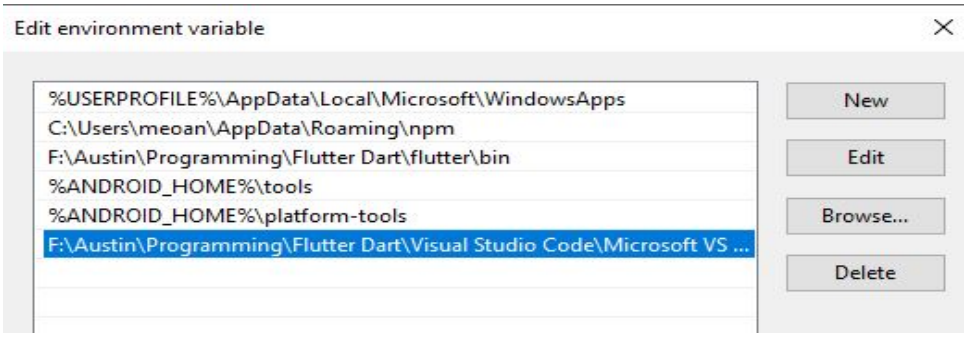
9. Click the "Environment Variables…" button.



10. Under User variables for <user>, click the Path variable, and then click the "Edit…" button.



11. Click the "New" button and enter the directory path to the Flutter SDK bin sub-folder you just installed.

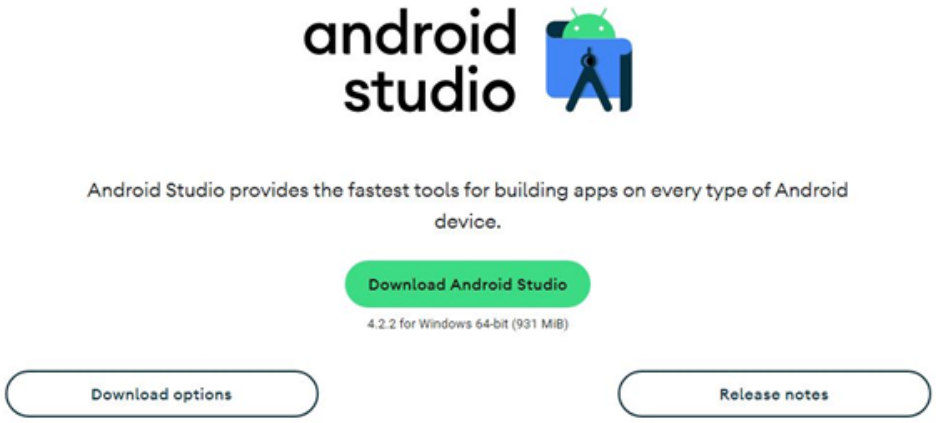


12. Open the command prompt and enter "flutter doctor" to confirm.

3.2 Installing Android Studio

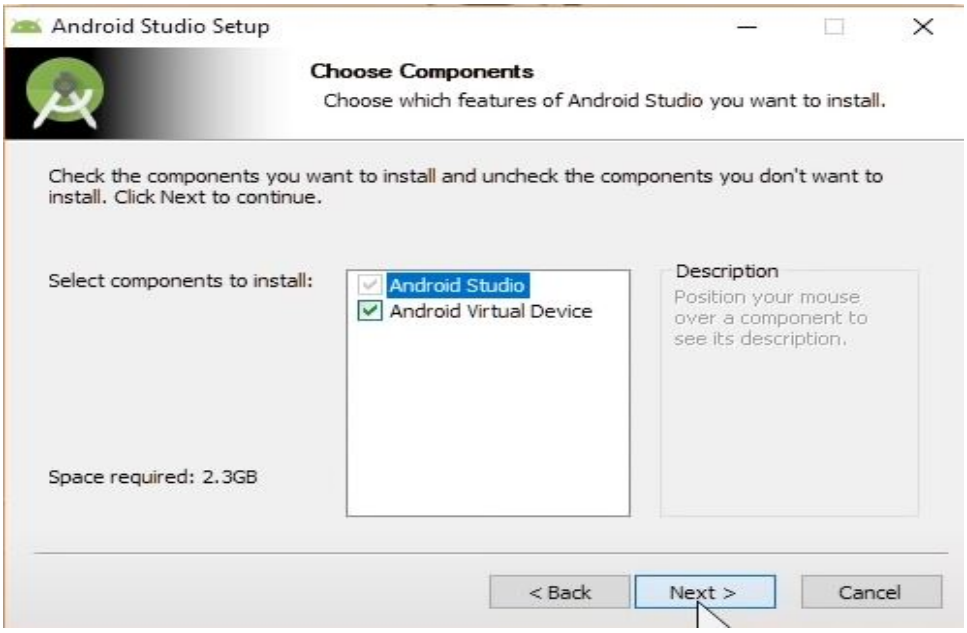
The following instructions assume a Windows environment, but the steps are largely the same for other supported operating systems.

1. Navigate to https://developer.android.com/studio
2. Click the "Download Android Studio" button.

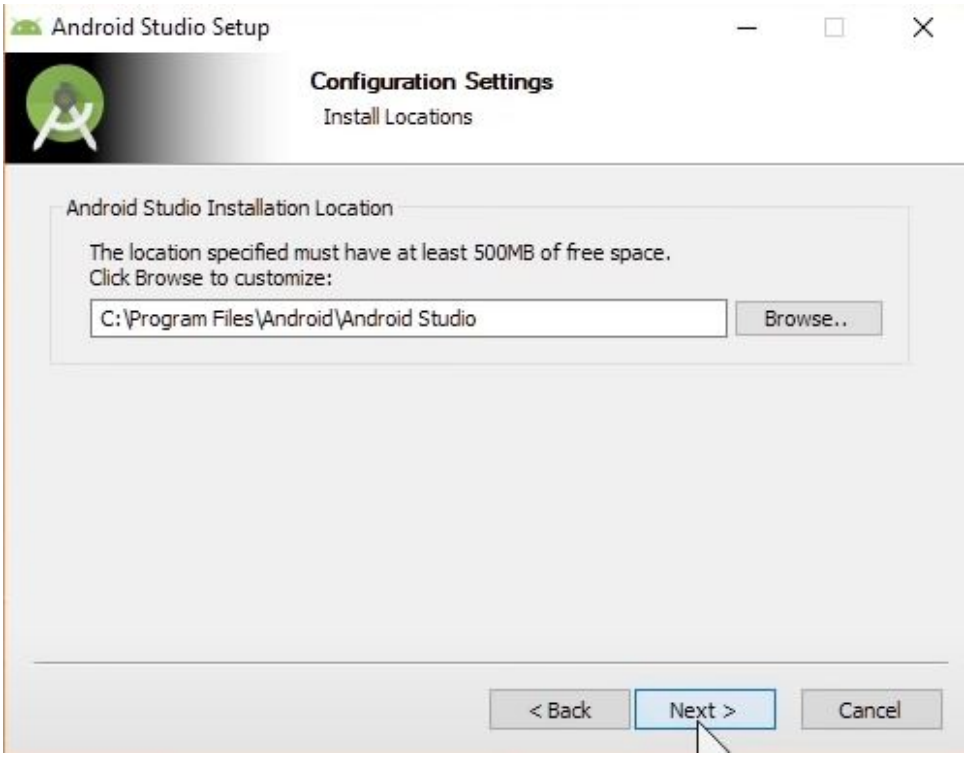


3. Accept the agreement and click "Download Android Studio for Windows."

4. Once downloaded, execute the exe file.
5. On the Choose Components screen, ensure that "Android Virtual Device" is checked.

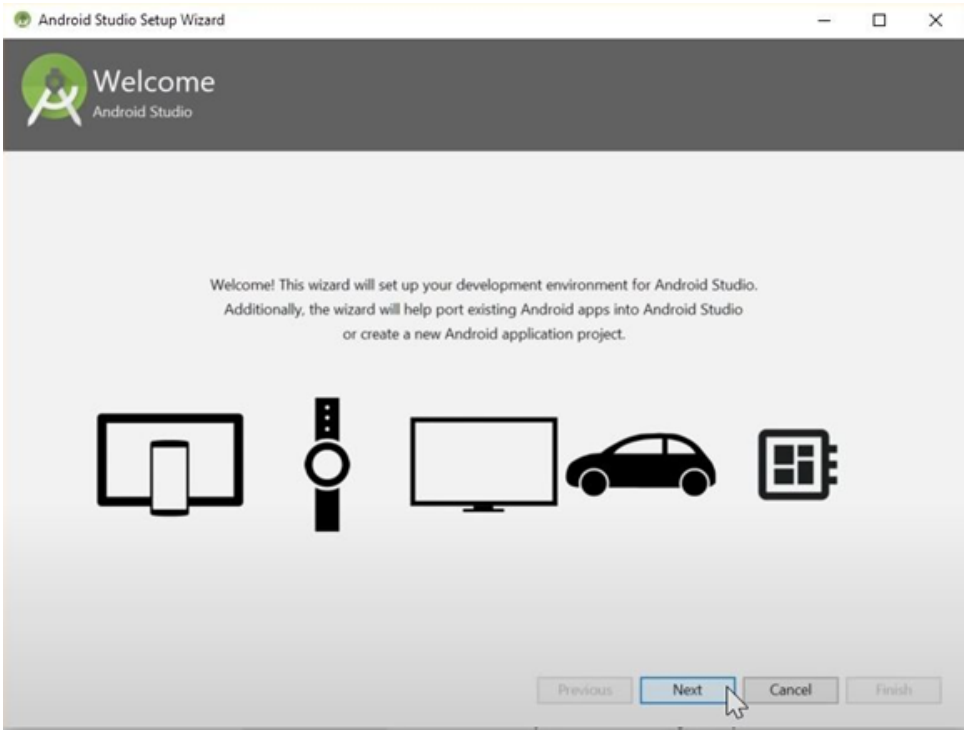


6. Click "Next >" until you get to the Installation Location and enter your desired location.

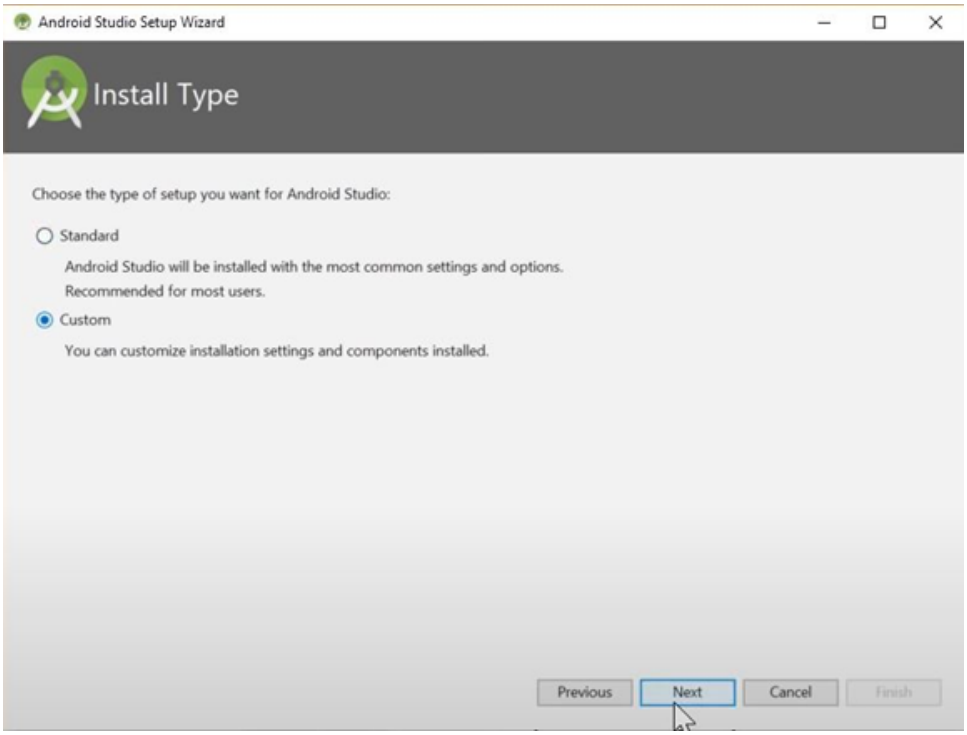


7. Click "Next >" until the application install.

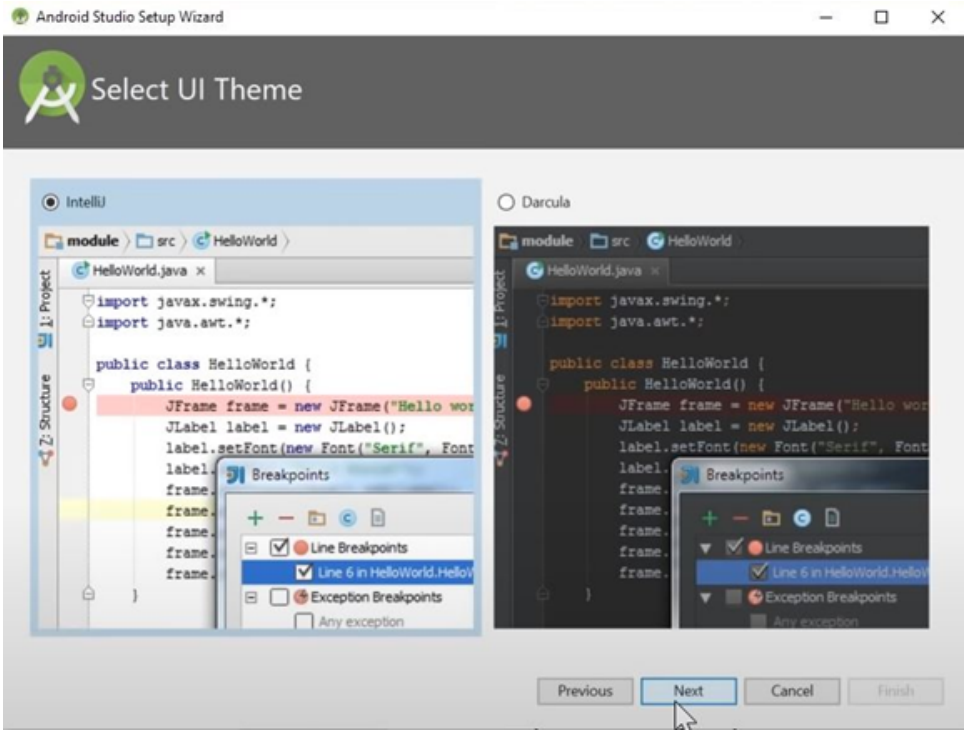
8. Start Android Studio and click "Next."



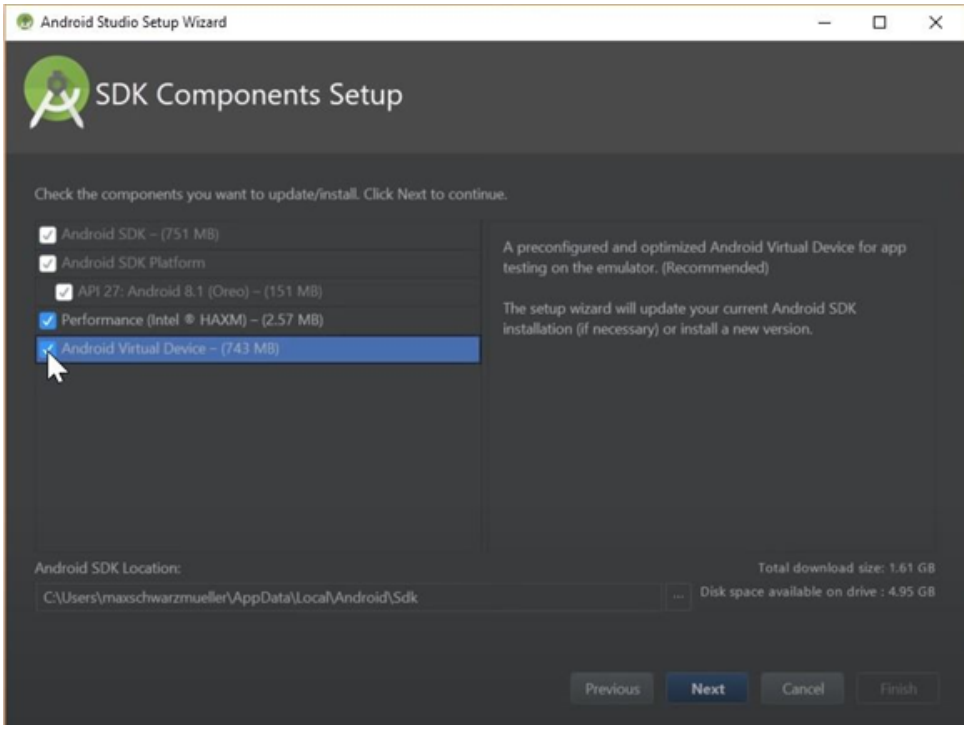
9. Click the "Custom" radio button and click "Next."



10. Select your desired UI Theme and click "Next."



11. In the SDL Components Setup screen, ensure that the "Android Virtual Device" box tick.



12. Click next and select the location that Android SDK install.

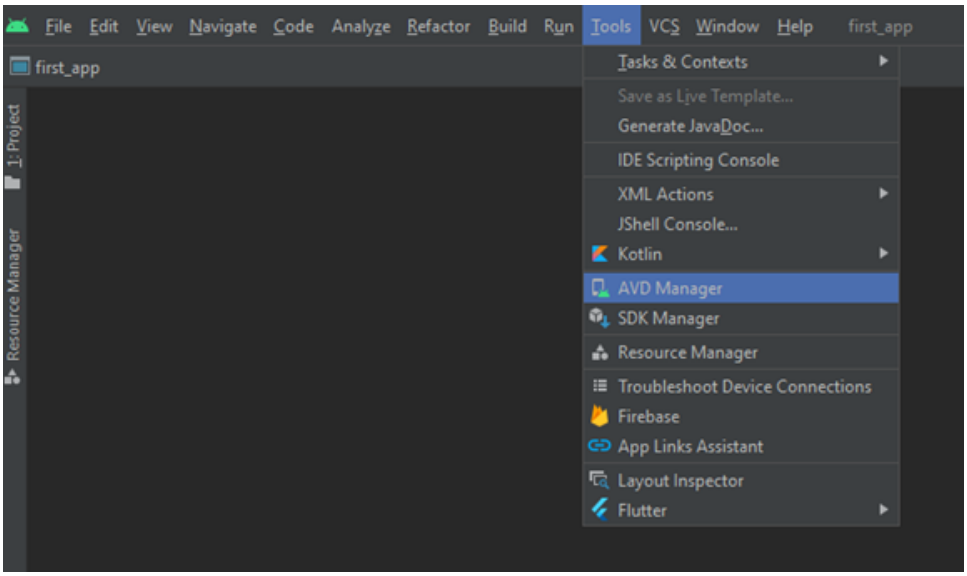
13. Click "Next" and then "Finish" to begin installing.

3.3 Setting Up an Android Emulator with Android Studio

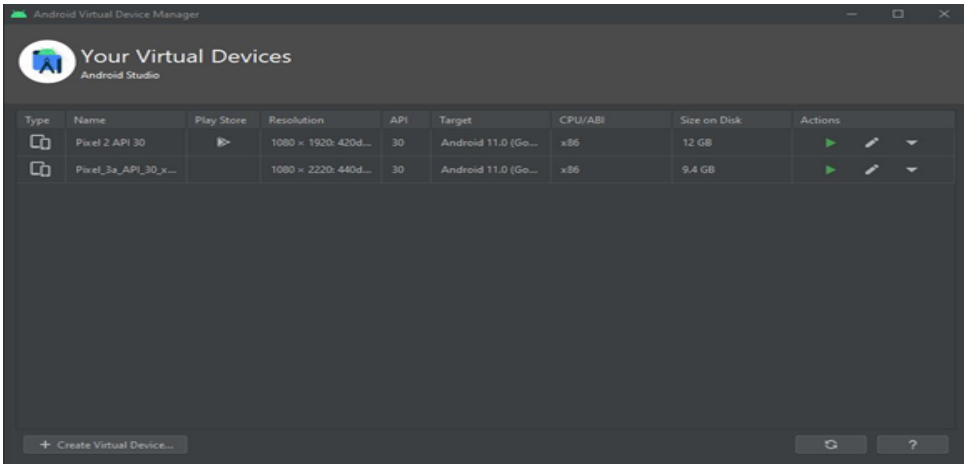
1. Enable VM acceleration on your machine.

2. Open Android Studio.

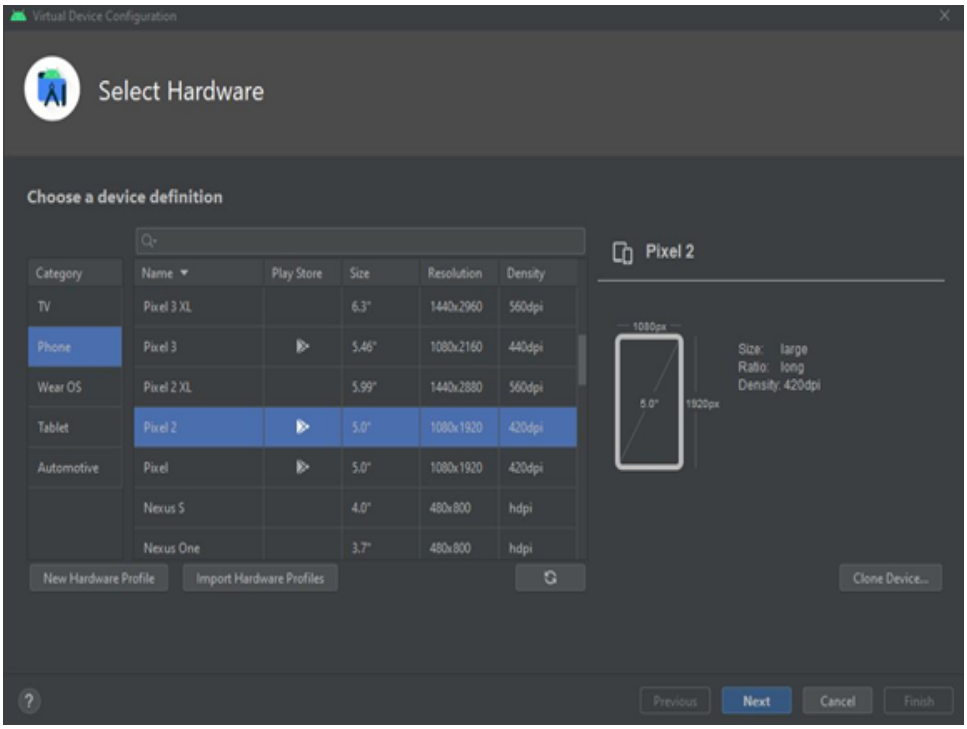
3. Click Tools > AVD Manager.



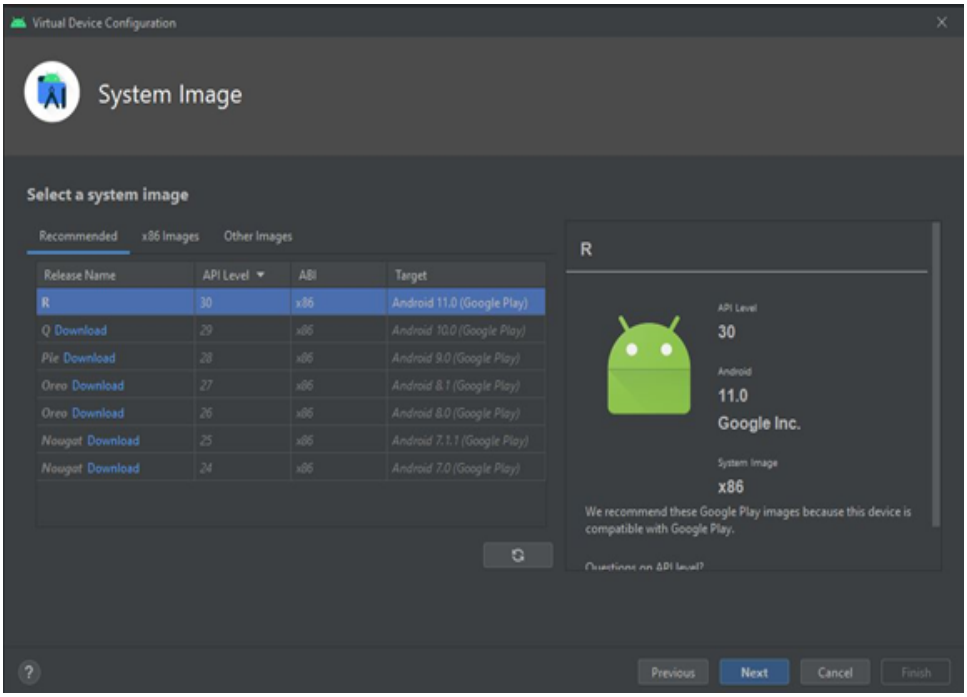
4. Click "Create Virtual Device…."



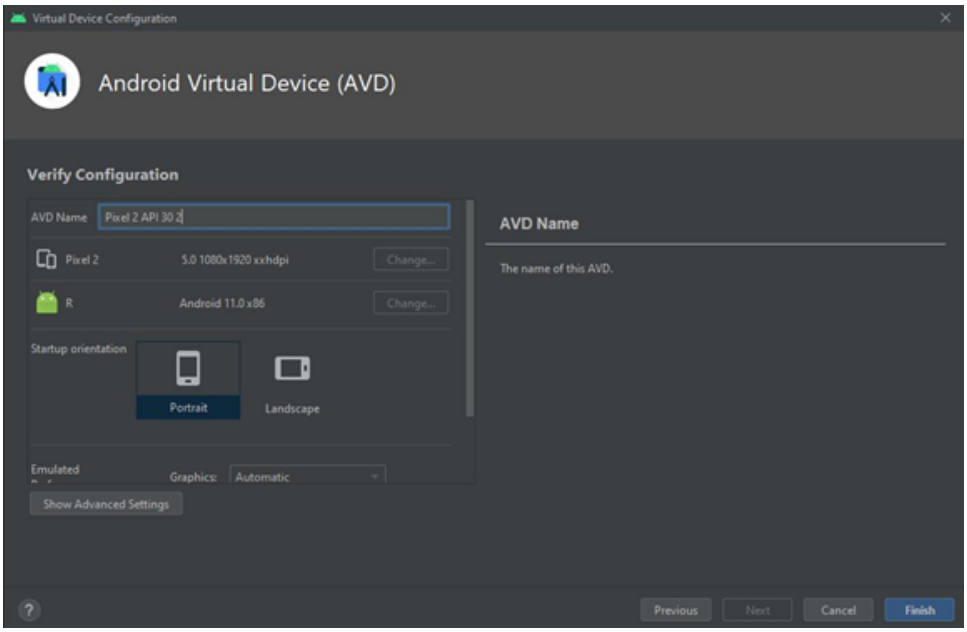
5. Select a device definition and click "Next."



6. Select a system image and click "Next."



7. Name the emulator and click "Finish."



8. To open this emulator from the command prompt, navigate to the emulator folder of your Android SDK installation, then run the command "emulator" followed by your emulator's name.

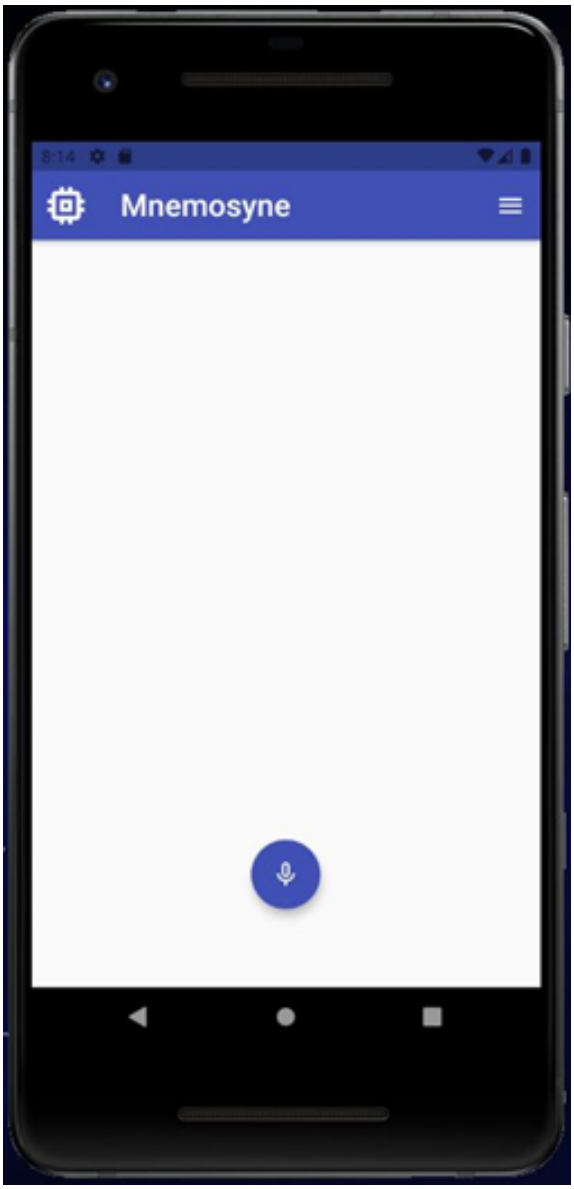
    For the emulator's name, start with an @ sign and use underscores for spaces.

3.4 Frontend Development

The application uses a modular design of pages that each serves a specific purpose. Four of these pages (Home, New, View, Settings) can be accessed through the hamburger menu at the top, while the final screen (Read Note screen) must access by navigating through the application. Because of this design, adding new features is relatively simple because new pages can just add to this system.

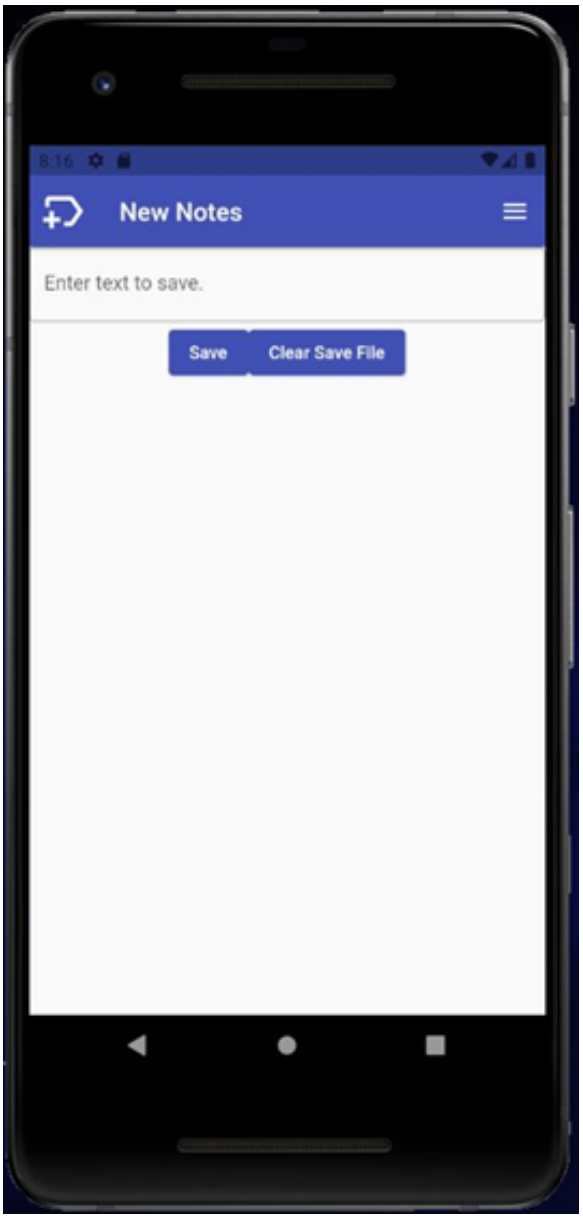
The screenshots below may be different from the final product depending on the development process. They included giving a reasonable representation of what the final app screen show.

3.4.1 Home Screen



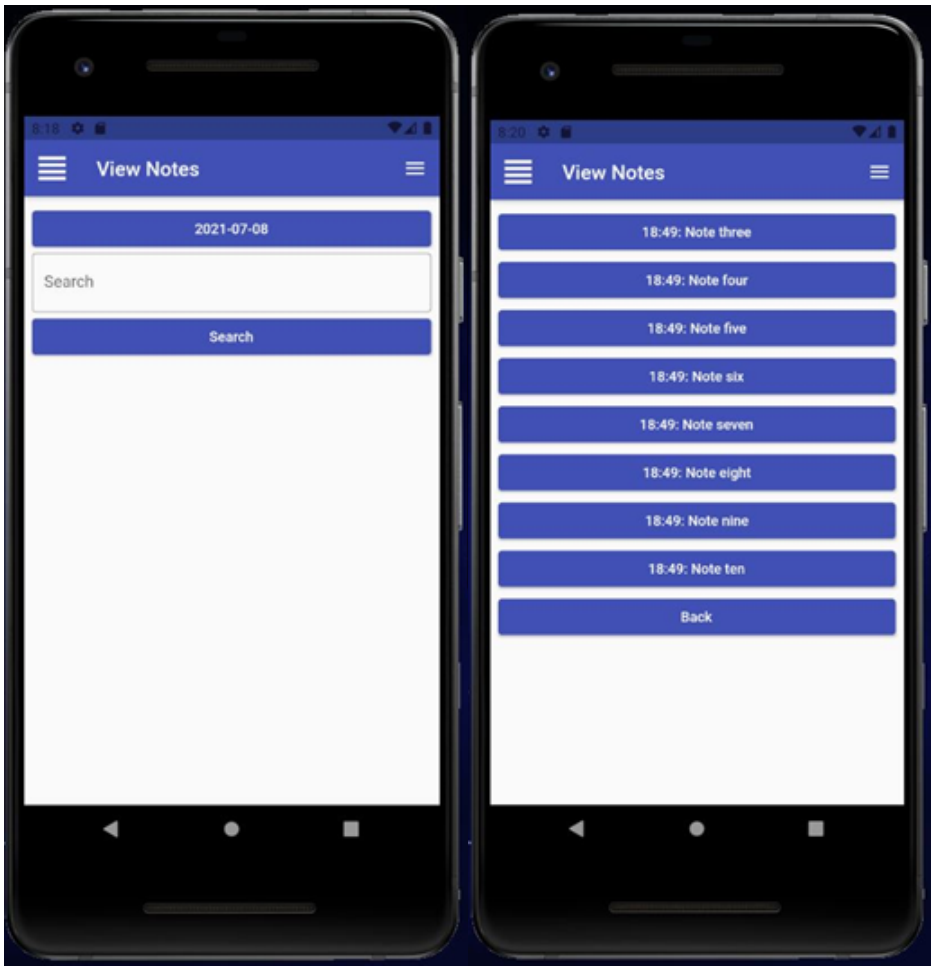
The Home screen is where the user starts, and it is where most of the speech functionality locate. By clicking the microphone button at the bottom of the screen, the app will begin listening to the user's microphone. When the user's voice is recognized, the words they are saying will display on the screen. When the recording end, the user's words save to a new note based on the date and time of when the button press.

3.4.2 New Note Screen



The New Note screen gives the user the option to manually add a new note by simply typing it out and clicking Save. The text entered will be saved as a new note at the date and time that it saves.

3.4.3 View Notes Screen



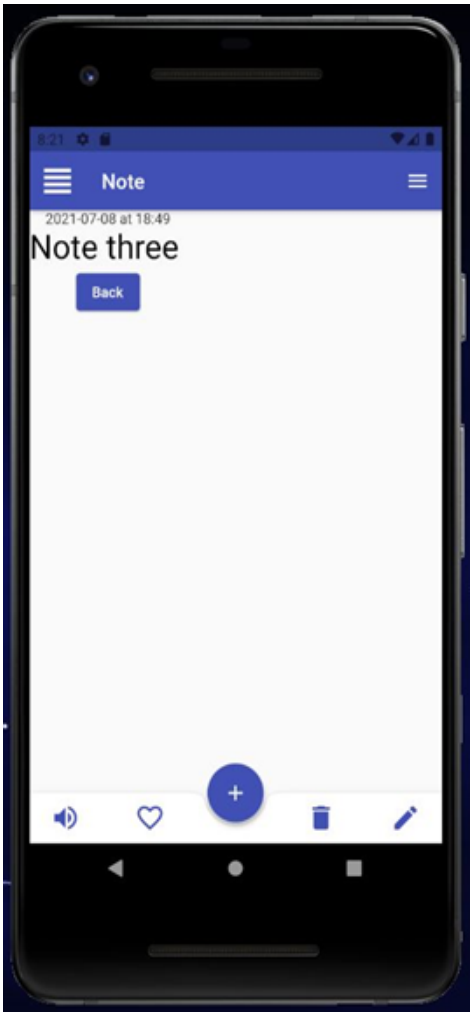
The View Notes screen is where the user can find previously saved notes. They are arranged by a series of button dates, which lead to a series of button times. Each time button represents a saved note, and a preview contents of that note display along with the time.

The search bar at the bottom of the list uses to enter a search phrase. If that phrase finds within any notes, they will list by date and time once the Search button is tapped. If the user taps the Clear Search button, the whole list show again.

3.4.4 Settings Screen

The Settings screen is where the user can perform various configuration tasks. These tasks include changing the appearance of zoom and boldness, training the app to the user's voice, configuring phrases to begin/endnote recordings, and viewing training videos that can teach the user how to use the app.

3.4.5 Read Note Screen



The Read Note screen is accessed by tapping one of the notes in the View Notes list. On this screen, the date and time note is at the top, followed by the actual contents of the note.

The Back button will return the user to the View Notes screen. The Plus button at the bottom will take the user to the Add Note screen. The Speaker button will read the note aloud using text to speech technology. The Heart button will enable whether or not the note is a favorited note or not. (Favorited notes require a fingerprint scan to access). The Trash button will delete the message. The Pencil button will take the user to a new screen to manually edit the note's contents.

3.5 Backend Development

The application's backend relies on the Flutter framework and Google cloud services for the application to function. The flutter framework and Dart programming language were used to develop the application and deploy it across Android and iOS platforms. The Google cloud service uses to stream audio to the cloud to convert it into text.

Most of the application simply takes advantage of Flutter features or public Flutter packages to function. These features can be utilized by any developer anywhere simply by using Flutter.

The speech-to-text features, however, rely on a Google cloud service. This service allows the application to stream audio to the cloud, convert it into text, and send it back to the application in real-time. The application can provide quick and reliable translations of voice into text that the user can see in real-time. Also, it means that the user needs some internet connection to record notes with their voice. The use of this cloud service requires a Google service account created by the developers. It is not an account the end-user needs or uses; it is simply a way for the application to connect to the Google service. The application automatically accesses this account to use the Google service, and the end-user never encounters it or has to configure it.

4 Code structure

As mentioned earlier, Flutter is a platform to build applications in all environments; mobile, desktop, and web. In this project, the platform uses mainly to build an app for mobile phones. The platform uses in Flutter is called Dart. Dart lets the programmers build the application with many components or modules. The Mnemosyne application consists of many such components, mainly classified into two categories: FrontEnd and BackEnd.

As the application is continues built iteratively, it is easy to have the code get piled up and difficult to read through later. As part of maintenance, it can lead to additional effort and cost to manage. It is critically important to keep the cord organized and easy to maintain.

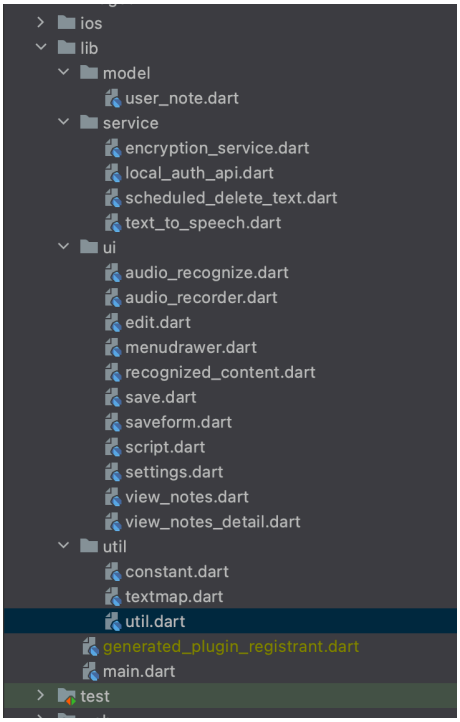


Figure 4.1 IDE code structure

As shown in Figure 4.1, the IDE, classes, or the dart files group to keep the type of files segregated. Each of the packages explains as follows:

- ui – Contains each of the dart files that create to establish widgets
- service – Contains the dart files that run in the backend as they trigger from the user interface or the background tasks that run on a schedule
- model – Contains the entity definitions
- util – Contains the utility classes that reuse throughout the application codebase.

The following sections will elaborate on each of the backend and frontend categories.

4.1 Frontend

In this application, the frontend part plays a significant role. The user interface is required to be self-intuitive and easy to navigate. Therefore, the components take the most advantage of their capability to create stateless and stateful widgets. The Mnemosyne app follows the concept of single page application. Moving from one page to other will not require a full- or partial-page refresh. The stateful widgets manage the component states without trailing between the server-side to the client-side. Following are the significant widgets that display in the application.

4.1.1 main.dart

Main Dart is the file where the application starts its operation. In the Mnemosyne application, the "main.dart" is a lightweight class that only initializes the application and calls the next page based on particular conditions and the work manager trigger.

```dart
/// callback dispatcher
void callbackDispatcher() {
  Workmanager().executeTask((task, inputData) async {
    var scheduledDeleteText = ScheduledDeleteText();
    scheduledDeleteText.deleteText(7);
    print("$simplePeriodic1HourTask was executed");

    ///Return true when the task executed successfully or not
    return Future.value(true);
  });
}

void main() {
  ///Initializing work manager
  WidgetsFlutterBinding.ensureInitialized();

  ///void initialize() {
  Workmanager().initialize(
    callbackDispatcher,
    isInDebugMode: true,
  );
  Workmanager().registerPeriodicTask(
    "6",
    simplePeriodic1HourTask,
    initialDelay: Duration(seconds: 10),
    frequency: Duration(minutes: 15),
  );

  runApp(MyApp());
}

///MyApp a stateless widget
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
```

Figure 4.1.1 main.dart file

As the application starts loading, two functions get triggered.

1. Loading the home page

2. Scheduled work-manager – jobs that need to run in a batch periodically

4.1.2 Home Page

In the case of this application, the application requires conditional home pages. As the application is installed and opened for a new user, the application requires a sample video to record to train the application with the user's voice. If the user has already completed this initial

step, then the user opens the application, it takes the user to the main page, where the user can start a recording. As seen in the [redacted] method definition, the application checks if the audio file exists. If the file exists, it directly takes the user to the audio recognize widget page. If the sample audio file does not exist, it takes the user to the audio recorder page.

```
58
59    Widget _getHome(context) {
60      return FutureBuilder<Widget>(
61          future: _getHomeWidget(),
62          builder: (context, snapshot) {
63            if (snapshot.hasData) {
64              return snapshot.data;
65            } else {
66              return CircularProgressIndicator();
67            }
68          });  // FutureBuilder
69    }
70
71    Future<Widget> _getHomeWidget() async {
72      if (await _audioFileExists) {
73        return AudioRecognize();
74      }
75      return AudioRecorder();
76    }
77
78    Future<bool> get _audioFileExists async {
79      final audioFilePath = await Constant.getAudioRecordingFilePath();
80      return Constant.getIfFileExists(audioFilePath);
81    }
82  }
```

Figure 4.1.2 Home page

4.1.3 audio_recorder.dart

The audio recorder is the first page that displays for a new user. It contains a widget to record the user's voice to save it as a sample wav file. The widget has a glowing elevated button that changes the icon from mic to stop based on the current state of the application. If the recording is in progress, the button displays the stop icon, otherwise the mic icon. As the user has completed the recording, clicking on the continue button takes the user to the audio recognize page.

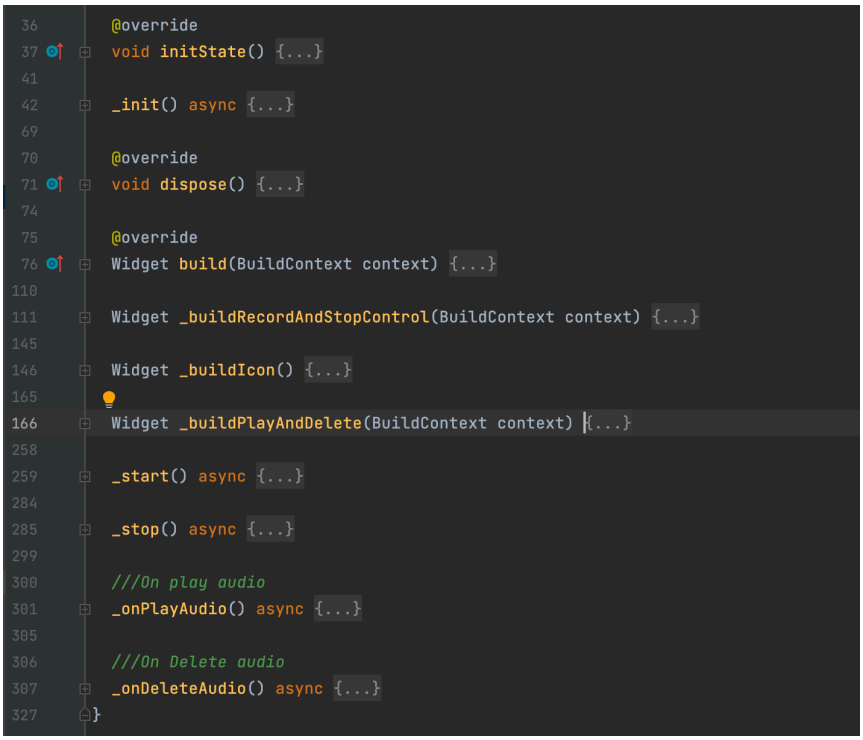


Figure 4.1.3 Audio_Recorder.dart

4.1.4 audio_recognize.dart

This dart file has a widget to display the mic, which changes the icon from mic to stop based on the current state. It streams the audio data from the device to Google's speech-to-text service.

Figure 4.1.4 Audio Recognize Widget

4.1.5 menu_drawer.dart

The hamburger menu at the top of the application draws the menu options to display the screens like view notes, edit the note, favorites, settings, and go back to the main audio recognize page. As can be seen, the [redacted] method builds the list title with the menu icons as mentioned in the method call. The method also accepts the callback method that gets triggered as the menus are tapped.

```
Widget build(BuildContext context) {
  return Drawer(
    child: ListView(
      padding: EdgeInsets.zero,
      children: <Widget>[
        Container(...),  // DrawerHeader, Container
        //Save Screen
        _buildListTitle(
          icon: Icons.add_circle_outline,
          title: "New Note",
          subTitleText: 'Type your notes to save along with other notes',
          subTitleFontColor: Colors.grey,
          onTapped: () {
            _saveNewNote(context);
          },
        ),
        //Load notes screen
        _buildListTitle(
          icon: Icons.note_alt_outlined,
          title: "View Notes",
          subTitleText: 'View all the saved notes',
          subTitleFontColor: Colors.grey,
          onTapped: () {
            _loadNotes(context);
          },
        ),
        //Favorites notes screen
        _buildListTitle(
          icon: Icons.favorite,
          title: "Personal/Favorites",
          subTitleText: 'View all personal or favorite notes',
          subTitleFontColor: Colors.grey,
          onTapped: () async {
            _loadFavorites(context);
```

Figure 4.1.5 Menu Drawer

4.1.6 view_notes_detail.dart

As the speech transcribes to text, the application receives the response from Google's speech-to-text API. The application saves the text in a file in the local memory of the application. When the user clicks the view note option in the menu drawer, it displays the notes in tiles, which click to view the text by date and time.

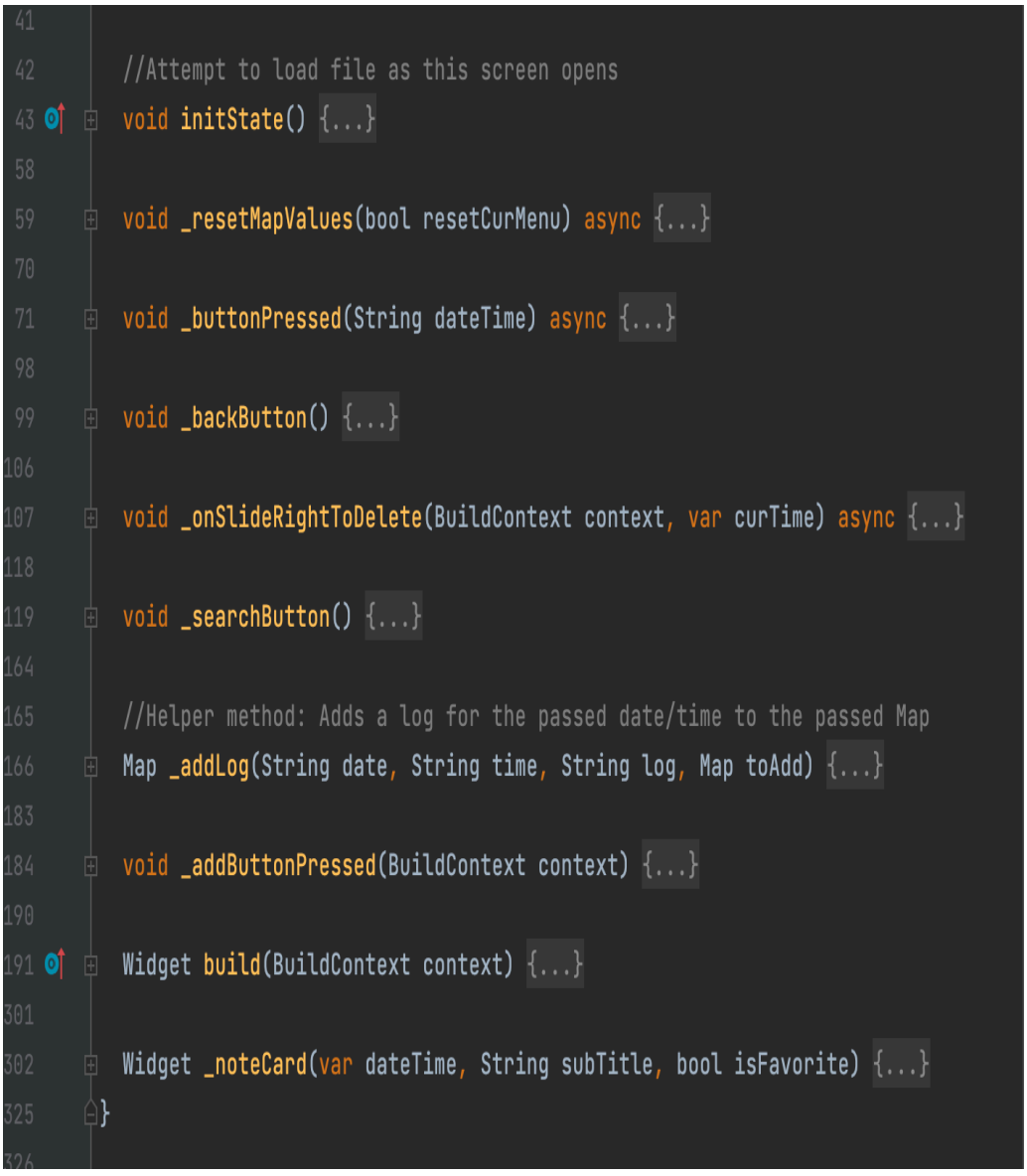
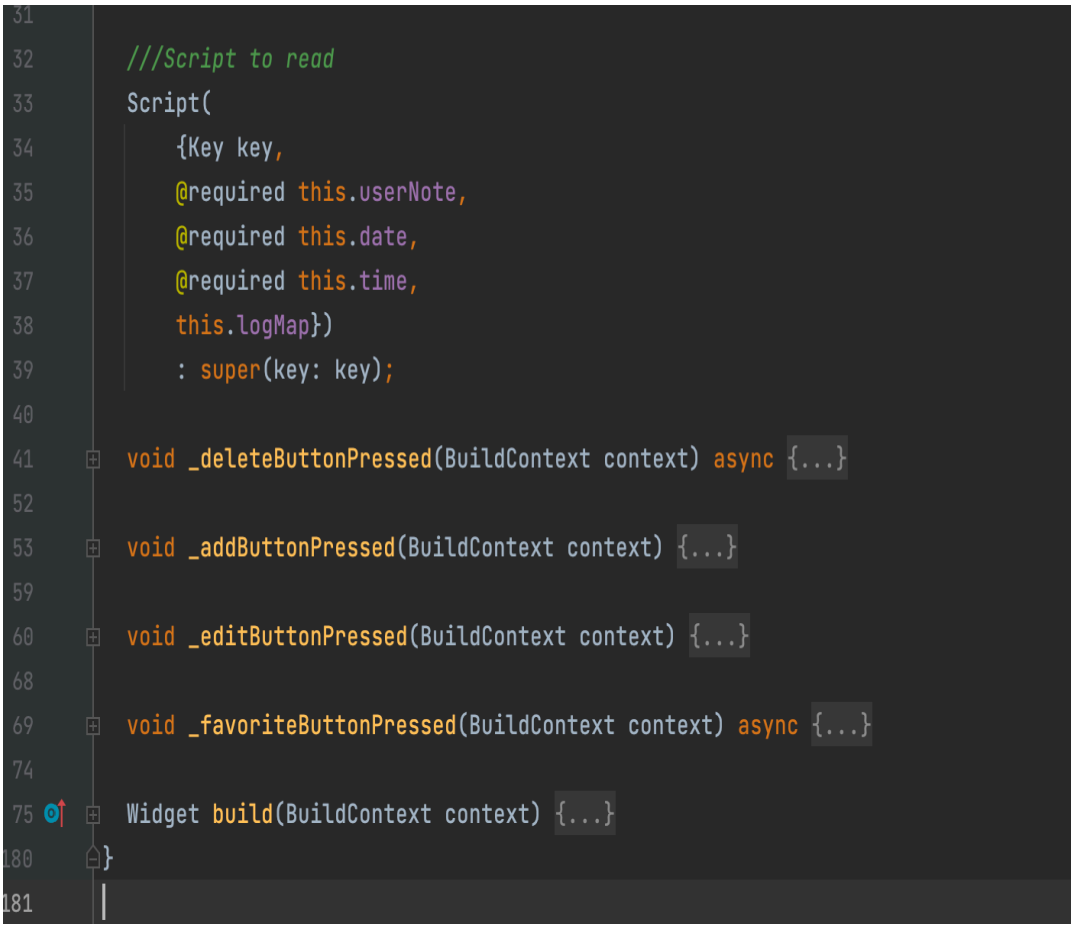


Figure 4.1.6 View Notes Details

4.1.7 script.dart

Script dart file displays the notes in a plain text format. The page has options to convert the text to speech, mark the script as favorite, Edit and delete the text. It also can take the user to the first page, where the user can start a new recording.



```
31
32      ///Script to read
33      Script(
34          {Key key,
35          @required this.userNote,
36          @required this.date,
37          @required this.time,
38          this.logMap})
39          : super(key: key);
40
41      void _deleteButtonPressed(BuildContext context) async {...}
52
53      void _addButtonPressed(BuildContext context) {...}
59
60      void _editButtonPressed(BuildContext context) {...}
68
69      void _favoriteButtonPressed(BuildContext context) async {...}
74
75      Widget build(BuildContext context) {...}
180 }
181
```

Figure 4.1.7 Script

4.2 Backend

The application has a few lightweight backend services. The below is the list of those services.

4.2.1 audio_recognize.dart

This service connects to Google's speech-to-text API via a service account. By using a service account, the speech is streamed to the API, and as the response, the text is received. This dart file contains both the front and backend part of the service.

```
101
102     final serviceAccount = ServiceAccount.fromString(...);
103     final speechToText = SpeechToText.viaServiceAccount(...);
104     final config = _getConfig();
105
106     final responseStream = speechToText.streamingRecognize(
107         StreamingRecognitionConfig(...),
108         _audioStream);
109
110     var responseText = '';
111
112     responseStream.listen((data) {
113       final currentText =
114           data.results.map((e) => e.alternatives.first.transcript).join('\n');
115
116       if (data.results.first.isFinal) {...} else {...}
129     }, onDone: () {
130       setState(() {
131         recognizing = false;
132       });
133     });
134   }
135
```

Figure 4.2.1 Audio_Recognize - speech to text

4.2.2 Speaker diarization service

As the requirement asks, the user's text is only supposed saved; the backend service takes advantage of speech diarization to mark the speakers. The recorded file is appended before the speech stream in every session that could be achievable. So, the service identifies the first voice as speaker #1.

Since the recorded voice belongs to the user, the converted text of speaker #1 only saves.

```
82    //Initializing the stream
83    _audioStream = BehaviorSubject<List<int>>();
84
85    //Getting the recorded audio file content and appending to the audiostream
86    var _fileStream = await _getAudioStream();
87    _fileStream.listen((event) {
88      _audioStream.add(event);
89    });
90
91    //Listening to the mic
92    _audioStreamSubscription = _recorder.audioStream.listen((event) {
93      _audioStream.add(event);
94    });
95
96    await _recorder.start();
97
98    setState(() {
99      recognizing = true;
100   });
101
```

Figure 4.2.2 Speak diarization service

4.2.3 text_to_speech.dart

This part is the service that converts the text to speech. It uses Flutter's tts (text-to-speech) library, which contains the plugins to convert.

```dart
///Text to speech service
class TextToSpeech {
  ///Speak method for text to speech
  Future speak(String text) async {
    final tts = FlutterTts();
    await tts.setLanguage('en-US');
    await tts.setPitch(1);
    await tts.speak(text);
  }
}
```

Figure 4.2.3 Text-to-Speech

4.2.4 encryption_service.dart

The encryption service uses Flutter's encrypt library. This service both encrypts and decrypts the text using a secret key.

```dart
  ///Initialization vector
  final iv = IV.fromLength(16);

  ///Encrypt
  String encrypt(String plainText) {
    final encryptor = Encrypter(AES(key));
    return encryptor.encrypt(plainText, iv: iv).base64;
  }

  ///Decrypt
  String decrypt(String encryptedBase64) {
    final encryptor = Encrypter(AES(key));
    return encryptedBase64.isNullOrEmpty()
        ? '{}'
        : encryptor.decrypt(Encrypted.from64(encryptedBase64), iv: iv);
  }
}
```

Figure 4.2.4 Encryption_Service

4.2.5 local_auth_api.dart

As part of the strategy to protect personal information, the local auth is used. As the users mark the favorites, they save with an isFavorite flag. When the user tries to view the notes saved as a favorite, the application prompts authenticating first. Flutters local_auth library has the implementation of the local authentication. It asks for biometric authentication as per the device. Also, it can be a fingerprint or a face recognition type. As shown in Figure 4.2.5, the favorites can view if the device has a biometrics authentication saved.

```dart
///Get biometrics
Future<List<BiometricType>> getBiometrics() async {
  try {
    return await _auth.getAvailableBiometrics();
  } on PlatformException catch (e) {
    print('Error in getBiometrics `$e');
    return <BiometricType>[];
  }
}

///Authenticate
static Future<bool> authenticate() async {
  final isAvailable = await hasBiometrics();
  if (!isAvailable) return false;

  try {
    return await _auth.authenticate(
      localizedReason: 'Scan Fingerprint to Authenticate',
      useErrorDialogs: true,
      stickyAuth: true,
      biometricOnly: true,
    );
  } on PlatformException catch (e) {
    print('Error in authenticate `$e');
    return false;
  }
}
}
```

Figure 4.2.5 Biometric authentication service

4.2.6 scheduled_delete_text.dart

The workmanager service runs in the background on an hourly basis and deletes the old texts except for the favorites. The current default threshold is seven days; however, it configures by the user.

```
/// delete
void deleteText(int numberDays) async {
  var fileText = await logs.getDecryptedContent();
  var dateTimeText = logs.readJson(input: fileText, filterFavorite: false);

  dateTimeText.removeWhere((key, value) {
    var dt = DateTime.parse(key);
    var data = json.decode(value);
    return dt.isBefore(DateTime.now().subtract(Duration(days: numberDays))) &&
        data['isFavorites'] == false;
  });
  //Write to file after adding log
  logs.writeFile(dateTimeText);
}
}
```

Figure 4.2.6 Scheduled delete text service

4.3 Dependencies

The project has many dependencies. The details of them are as follows.

- google_speech: ^2.0.1 – Used for speech-to-text transcription
- sound_stream: ^0.2.0 – To stream mic audio to external services
- rxdart: ^0.27.0 – Use streaming services efficiently
- highlight_text: ^0.7.2 – To display the saved vocabulary texts in a highlighted format
- avatar_glow: ^1.2.0 – To display a glow effect on the button when it is tapped/pressed
- workmanager: ^0.3.0 – Used to run batch jobs in the background
- encrypt: ^5.0.0 – Used to encrypt or decrypt text
- flutter_tts: ^3.1.0 – To convert text to speech

- path_provider: ^2.0.1 – Used to navigate to different paths in the app memory
- sound_recorder: ^0.0.2 – A library that provides utility methods to record user's voice
- audioplayers: ^0.18.3 – Used to display an audio player to playback the recorded audio
- local_auth: ^1.1.6 – Used for biometrics authentication

```
environment:
  sdk: ">=2.7.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter
  google_speech: ^2.0.1
  sound_stream: ^0.2.0
  rxdart: ^0.27.0
  highlight_text: ^0.7.2
  avatar_glow: ^1.2.0
  workmanager: ^0.3.0
  encrypt: ^5.0.0
  flutter_tts: ^3.1.0
  path_provider: ^2.0.1
  sound_recorder: ^0.0.2
  audioplayers: ^0.18.3
  local_auth: ^1.1.6


  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.3
```