# 4 Code structure

Graphical user interface, text

Description automatically generated with medium confidenceAs mentioned earlier, Flutter is a platform to build applications in all environments; mobile, desktop and web. In this project the platform is used mainly to build an app in mobile phone. The platform that is used 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 which can be mainly classified into two categories: FrontEnd and BackEnd.

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

As can be seen in Figure 4.1, in the IDE the classes or the dart files have been grouped to keep the type of files in a segregated way. Each of the package can be explained as following.

* ui – Contains each of the dart files that are created to create widgets
* service – Contains the dart files that run in the backend as they are triggered from the user interface or the background tasks that are run on a schedule
* model – Contains the entity definitions
* util – Contains the utility classes that can be reused throughout the application codebase.

Figure 4.1 IDE code structure

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

## 4.1 Frontend

In this application the frontend part plays the major role. The user interface is required to be self-intuitive and easy to navigate. Therefore, the components take the most advantage of its 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 states of the component without having to traverse between the server side to client side. Following are the major widgets that display in the application.

## 4.1.2 main.dart

Main dart is the file where the application starts its operation. In the Mnemosyne application, the “main.dart” is a lightweight class which only initializes the application and calls the next page based on certain condition and the work manager trigger.

Text

Description automatically generated

Figure 4.1.1 main.dart file

As the applications starts loading two functions get triggered.

1. Loading the home page
2. Scheduled work-manager – jobs that need to be run in a batch periodically

## 4.1.2 Home Page

In case of this application, application requires home pages that are conditional. For a new user, as the application is installed and opened, the application requires a sample video that needs to be recorded to train the application with the user’s voice. If the user has already completed this initial step, then as the user opens the application, it takes the user to the main page where the user can start a recording. As can be seen in the \_getHomeWidget() 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, then it takes the user to audio recorder page.

Text

Description automatically generated

Figure 4.1.2 Home page

## 4.1.3 audio\_recorder.dart

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

Text

Description automatically generated

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 device to Google’s speech-to-text service.

Text

Description automatically generated

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 note, edit note, favorites, settings and go back to main audio recognize page. As can be seen the \_buildListTitle() 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.

Text

Description automatically generated

Figure 4.1.5 Menu Drawer

## 4.1.6 view\_notes\_detail.dart

As the speech is transcribed 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 user clicks view note option in the menu drawer, it displays the notes in tiles which can be clicked to view the text by date and time.

Text

Description automatically generated

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 has an option to take the user to the first page where user can start a new recording.

**Text

Description automatically generated**

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

Text

Description automatically generated

Figure 4.2.1 Audio\_Recognize - speech to text

## 4.2.2 Speaker diarization service

Text

Description automatically generatedAs the requirement asks, the text of the user is only supposed to be saved, the backend service takes the advantage of speech diarization to mark the speakers. To achieve it, the recorded file is appended before the speech stream in every session. So, the service identifies the first voice as the speaker #1. Since the recorded voice belongs to the user, the converted text of the speaker #1 is only saved.

## 4.2.3 text\_to\_speech.dart

This is a service that converts the text to speech. This uses the flutter’s tts (text-to-speech) library, which contains the plugins to convert.

Text

Description automatically generated

Figure 4.2.3 Text-to-Speech

## 4.2.4 encryption\_service.dart

Encryption service uses flutter’s encrypt library. This service both encrypts and decrypts the text using a secret key.

Text

Description automatically generated

Figure 4.2.4 Encryption\_Service

## 4.2.5 local\_auth\_api.dart

As part of the strategy to protect the personal information, the local auth is used. As the users mark the favorites, they will be saved with an isFavorite flag. When user tries to view the notes that are saved as favorite, then the application prompts to authenticate first. Flutters local\_auth library has the implementation of the local authentication. It asks for the biometric authentication as per the device. It can be a fingerprint or a face recognition type. As can be seen in the figure 4.2.5, the favorites can be viewed only if the device has a biometrics authentication saved.

Text

Description automatically generated

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 the favorites. The current default threshold is 7 days; however, it can be configured by the user.

Text

Description automatically generated

Figure 4.2.6 Scheduled delete text service