Language Identification Software **Technical Manual**

Version 1

17 August 2023 (Update this)

Jordan Wylde-Browne, Toby Coy, Leo Headley,

Thomas Ambrose, Theodore Tiong, Brayden Ransom-Frost

Table of Contents

1. Application Structure & Architecture

There are two main Flutter applications, the main application and the daughter app. The main application is used for language identification, audio playback, and answer recording. The daughter app is intended to allow for easier modification of the database which manages the questions & statements and audio.

1.1. Main Application

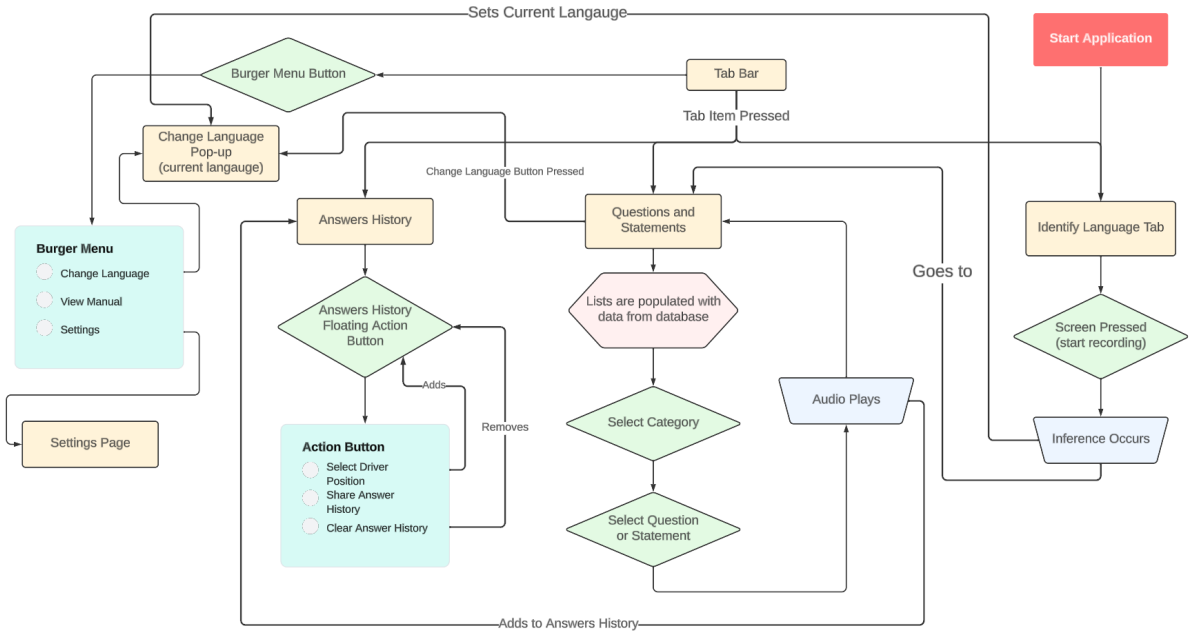


Figure 1. User flow of the main application.

As shown in figure 1, the user flow of the application can be quite complex. It begins at the invitation to speak screen, which is the expected start point for all use cases. Users can press the screen to record audio, and then it will predict the language spoken. When a language has been identified, questions can be asked through the questions and statements screen, which play audio back in the identified language. Any questions that prompt responses will have answers saved into some global state, and these can be viewed in the answers history page. The settings page, accessed by the burger menu, allows for customization of some functionality in the app as well as various utilities.

1.1.1. Major Components

The app is split into three main pages, all controlled through a central tab controller in *main.dart*. The app sits under all the providers used for managing global state, such as the categories, language, and various utilities. These providers are then passed into individual pages and widgets as needed. Other components of the application are accessed through either the burger menu or various buttons throughout the app.

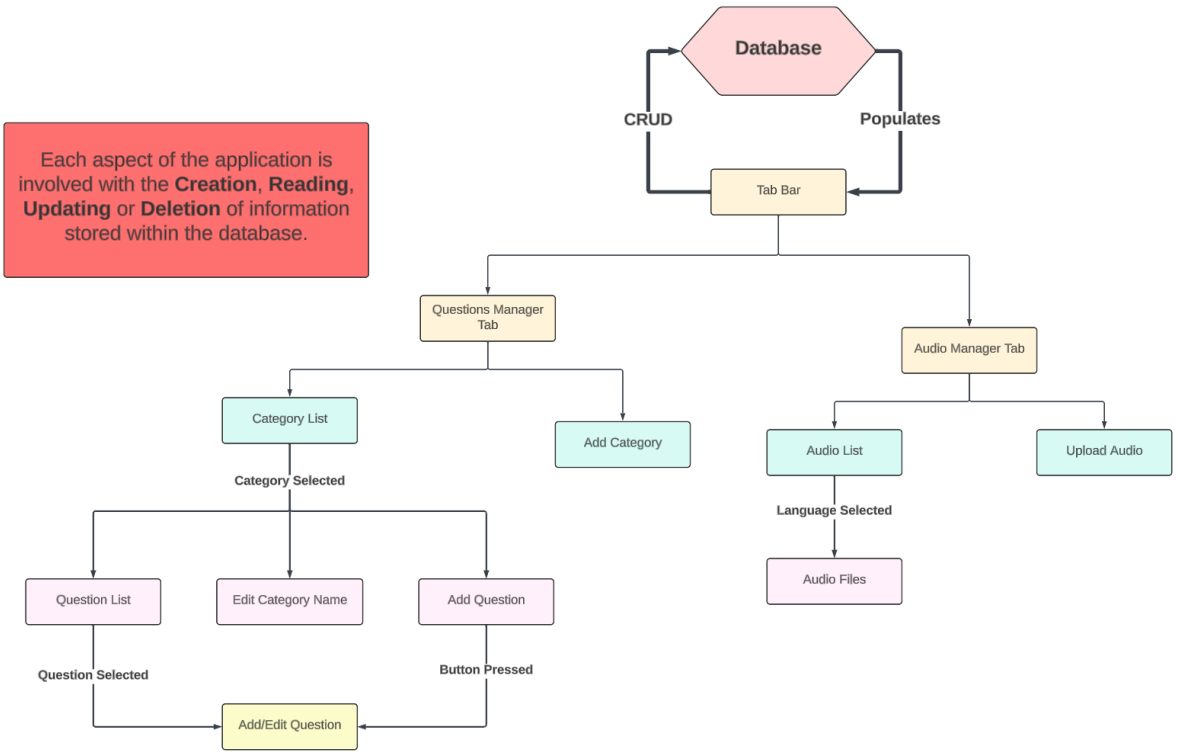
Invitation to Speak
This screen contains the audio recorder and animation. When the user presses any part of the screen, audio recording starts, and the animation changes state. When the recording is finished, language inference is completed on the saved recording, and the global language state is updated. This screen has no state of its own, but manages the state of the answers and language providers.

Questions & Statements
The questions screen displays a list of categories and a list of questions under that categories. These are loaded from the categories provider, which is initialised at the start of runtime. This screen has a lot of state to manage, as the list of questions is based on the currently selected question. Additionally, it manages audio playback within the same screen. It must communicate and interface with most providers in the application. The user is only able to directly modify the state of the viewable questions by either changing the language, or downloading new ones through the settings page.

Answers History
The answers tab exists to display the list of answers from the answers provider, as well as giving the user the avenue for managing this state, such as clearing the list or deleting answers. The screen itself manages almost no state of its own, and is primarily there to act as a view for the answers model.

Settings
The settings page communicates with a number of providers as it gives the user the ability to update and modify state in the application. It allows users to download audio and questions from the database, saving the files to the device, as well as changing colour themes, and modifying parts of the functionality of the app. It is built as a list of components which each provide different functions and widget types, and is built on top of the *settings_ui* package.

1.2. Daughter Application



2. Database Structure

Database tables? Talk about firebase here

3. Application Source Code

Filename	main.dart
Purpose	The main dart file contains the driver and main widget for the application, which is a tab view. This is the start point for the application.
Classes, Functions & Content	***main()*** Drives the app and starts runtime, also initialises Firebase as well as some utilities. **MyApp** Builds *MyHomePage* with the various providers needed. **MyHomePage** The main application widget contains the tab bar controller and initialises state for user preferences. **NavigationTab** A custom styled tab for the navigation tabs in the top bar. **BurgerMenu**

	Creates a burger menu widget to be placed at the top of the tab bar, leading to various utilities in the application.

3.1. Main Pages

Filename	invitation_tab.dart
Purpose	The invitation tab file contains the main widget for this screen. It is used to display an animation and record audio, then predict the language spoken.
Classes & Content	**InvitationTab** Contains all state for the invitation to speak page, and renders an *AudioRecorder* widget as its primary view.

Filename	questions_tab.dart
Purpose	The questions tab dart file contains all relevant widgets to the questions and statements screen, including question widgets, the main screen, and all dialogs. It is used to access a large array of questions to play audio for.
Classes & Content	**QuestionsTab** Contains all state for the questions & statements page, rendering two lists displaying all questions and statements from the categories model. Also contains state and logic for playing audio from the questions list. **YesNoDialog** Renders a dialog to display the options to press yes or no in response to a question. **ScaleRatingDialog** Renders a dialog to display a scale rating from 1 - 10 in response to a question. **MultipleChoiceDialog** Renders a dialog to display a multiple choice answer in response to a question. **CustomSliderThumbCircle** Custom styled slider thumb used for the scale rating dialog. **ConfirmationDialog** Renders a dialog for confirming audio playback, manages some state for the question pressed.

Filename	answers_tab.dart
Purpose	This file contains all relevant widgets to the answers history screen, including FABs, and the screen itself. It is used to view a history of answers after asking questions.
Classes & Content	**AnswersTab** Contains all state for the answers history page, and renders the list of answers from the provider.

	FabWithIcons Creates an extended floating action button which is used for various utilities on the screen. **SeatPositionDialog** Renders a dialog for modifying the answers provider and selecting a seat position.

Filename	settings_screen.dart
Purpose	This dart file contains all relevant widgets to the settings screen, including dialogs and the screen itself. It is used to manage personal settings throughout the app and access utility features.
Classes & Content	**DownloadButton** A small styled button widget for downloading items. **Section** A custom section for a settings list, styled as needed. This extends an AbstractSettingsSection from the *settings_ui* package. **SettingsWidget** The entire settings screen contains a list of settings and all buttons and functions associated with updating user preferences.

3.2. Utility & Miscellaneous

Filename	audio_recorder.dart
Purpose	The audio recorder file contains widgets for managing the audio recorder used in *invitation_tab.dart*. It is used to record audio and then save it to the device.
Classes & Content	**AudioRecorder** The widget for the audio recorder, containing all states for both the animation and the audio recording.

Filename	log.dart
Purpose	Small utility file extending a custom Logger for debugging.
Classes & Content	**Log** Extends a debugger with some custom styling.

Filename	pdf_viewer.dart
Purpose	The PDF viewer contains a widget for managing a pdf viewer, primarily used for viewing the user manual in the application.
Classes &	**PDFViewerFromAsset**

Content	Loads a PDF from assets then displays it in a reader on screen as a widget.

3.3. Language Inference

Filename	ml_inference.dart
Purpose	This file contains no widgets, but instead a number of functions used for language inference in *invitation_tab.dart*.
Functions & Content	***predictLanguage(String)*** Loads audio, transforms it into a spectrogram, then completes inference. ***loadAudio(String)*** Loads audio given a path, deletes the file, then returns the signal. ***inference(Matrix)*** Loads the TensorFlow interpreter and then completes inference given an input signal matrix.

Filename	spectrogram.dart
Purpose	This file contains no widgets, but instead a large number of functions for processing audio into spectrograms ready for language inference in *ml_inference.dart*.
Functions & Content	***msFrames(int, int)*** Converts ms to number of frames given some sample rate and ms ***hertzToMel(double[])*** Maps hertz frequencies to a mel scale. ***cmvn(Matrix)*** Computes the central mean variance normalisation of a given matrix. ***stft(double[], fn(), int, int, int, float[])*** Computes the short time fourier transform for a given input signal. ***computeMelWeightsMatrix(int, int, int, double, double)*** Computes a mel weights matrix given some parameters. ***melSpectrogram(double[])*** Creates and returns a log mel spectrogram given some signal. ***removeSilence(double[], int)*** Removes silence from a signal using framewise VAD. ***nonOverlapFrame(double[], int)*** Transforms an input signal into non overlapping frames. ***rootMeanSquare(double[][], int)*** Computes the root mean square of an input. ***invertShortConesecutive(bool[], int)*** Used by VAD decisions to invert binary decisions. ***framewiseVAD(double[], int, int, int, int, double, double, int)*** Computes framewise VAD decisions for an input signal for noise filtering.

3.4. Providers

The following are all files that contain change notifier providers, used to maintain some global state across the application.

Filename	answers.dart
Purpose	The answers provider maintains state on the answers history and any answers to question dialogs.
Classes & Content	**Answer** Abstracts information for an answer, including the question asked and the answer given. **AnswersModel** The provider for the answers model.

Filename	audio_downloader.dart
Purpose	The audio downloader provider is used to maintain downloading state while downloading audio from the database.
Classes & Content	**AudioDownloader** The provider for the audio downloader.

Filename	category.dart
Purpose	The category provider manages the list of categories and questions, as well as any state associated with loading these, from the database or from local files.
Classes & Content	**Category** Abstracts information for a category, including the name and list of questions. **Question** Abstracts information for a question, including the name, id, and question itself. **CategoriesModel** The provider for the category model.

Filename	themes.dart
Purpose	The themes provider manages state on the colour themes used throughout the application, as well as the currently selected theme.
Classes & Content	**ThemeModel** The provider for the theme model. **AppThemeData** Wrapper of ThemeData class to give themes a name.

Filename	language.dart
Purpose	The language provider manages state and interfaces with the language list used in the application. It also contains widgets for the language dialog.
Classes & Content	**LanguageModel** The provider for the language model. **LanguageDialog** The dialog for changing the language.

4. Development Tools & Dependencies

4.1. Development Tools

Name	Python
Type	Programming Language
Version	TODO
Purpose	Python is the primary language used for machine learning, it is object-oriented and popular for this use case.

Name	Flutter
Type	Software Development Kit
Version	>=3.7.0
Purpose	Flutter is a software development framework created by Google used to create cross-platform applications.

Name	Dart
Type	Programming Language
Version	>=2.19.5 < 3.0.0
Purpose	Dart is the primary language used by the Flutter framework.

Name	Visual Studio Code
Type	Text Editor
Version	TODO
Purpose	Visual Studio Code, created by Microsoft, is the primary code editor used for the Flutter & Python development.

Name	Android Studio
Type	Integrated Development Environment
Version	TODO
Purpose	Android Studio is the official development environment for Android development and was used by some team members for Flutter development.

4.2. Python Dependencies

TODO

4.3. Flutter Dependencies

Name	Logger
Type	Flutter Plugin
Version	1.4.0
Purpose	A logging framework used to debug the application during development.

Name	Share Plus
Type	Flutter Plugin
Version	7.1.0
Purpose	A simple plugin for creating dialogs for sharing information across devices.

Name	Cloud Firestore
Type	Flutter Plugin
Version	4.8.2
Purpose	A plugin used to maintain, connect to, and communicate with cloud storage through Firebase.

Name	Firebase Storage
Type	Flutter Plugin
Version	11.2.6

Purpose	A plugin used to maintain, connect to, and communicate with a Firebase database.

Name	Flutter Sound
Type	Flutter Plugin
Version	9.2.13
Purpose	Plugin used for audio recording and manipulation.

Name	Audioplayers
Type	Flutter Plugin
Version	4.1.0
Purpose	Plugin used for audio playback.

Name	FFTea
Type	Flutter Plugin
Version	1.3.1
Purpose	FFTea is a plugin which runs optimised Fast Fourier Transforms, functions used when converting the audio for inference.

Name	Ml Linalg
Type	Flutter Plugin
Version	13.11.31
Purpose	Utility package used for its linear algebra functions.

Name	Scidart
Type	Flutter Plugin
Version	0.0.2-dev.12
Purpose	Utility package used for its linear algebra, as well as FFT helper functions.

Name	Wav
Type	Flutter Plugin

Version	1.3.0
Purpose	Simple package which allows for WAV file manipulation.

Name	Tflite Flutter
Type	Flutter Plugin
Version	0.10.1
Purpose	Allows for loading and usage of TFLite models for running machine learning inference on devices.

Name	Settings UI
Type	Flutter Plugin
Version	2.0.2
Purpose	A modular settings page builder plugin, used for building a custom settings list.

Name	Flutter PDFview
Type	Flutter Plugin
Version	1.3.1
Purpose	Simple implementation of a PDF viewer within a Flutter application.

Name	Permission Handler
Type	Flutter Plugin
Version	10.3.0
Purpose	Handles permissions for audio recording and file handling.

Name	Rive
Type	Flutter Plugin
Version	0.11.4
Purpose	An animation framework, used primarily for the invitation to speak screen animation.

Name	Flutter Launcher Icons

Type	Flutter Plugin
Version	0.13.1
Purpose	Used to create custom icons for the application.

Name	Shared Preferences
Type	Flutter Plugin
Version	2.2.1
Purpose	State management and user preferences plugin used to remember settings across uses.

5. Updating the Application

The scale of the project is large and has many components to it. The following is some general guidelines on updating the application and modifying it for future use.

5.1. Application Updates

The application is built with the Flutter SDK, all relevant versions for Dart and Flutter can be found in Section 4.1. The project can be set up as standard for Flutter projects.
The app will not run on web builds, due to the drivers used for machine learning. It has been primarily tested and run on a Nexus 9 API 34 emulator.
When the project is set up, it can be built and run on the device, and changes can be made.

Changes to the questions & statements, and audio, can be managed through the daughter application, or directly through the Firebase database web interface. When files are downloaded through settings they are loaded into a local cache, these can be deleted or removed as needed. Questions & statements are loaded as json files, and audio as mp3.

The available languages are managed through the *assets/ml/label_maps.json* file, and loaded in *language.dart*, which is simply an array of languages with their code and full title. Note that the order of these languages does matter, as the language model will use a simple index to identify the language from the list, so it must match the order of the languages the model has been trained on.

Additionally, under the *assets/ml/* folder is the currently used language model. Any number of models can be added here, however only the one specified in *ml_inference.dart* will be used.

The user manual is found under *assets/pdf/user_manual.pdf*. It is specified in *pdf_viewer.dart*, this can be changed there.

5.1.1. Tensorflow Updates

Note that the application uses machine learning on-device using its GPU. This requires special installation of drivers to ensure the device can correctly use Tensorflow. In the project, the drivers should be seen under *android/app/src/main/jniLibs/*. If they are not, then run the installation script found under the root of the project.

5.2. Language Model

5.2.1. Training a model

TODO

5.3. Database Updates

-firebase login details (?)
-how to change through interface
-basic explanation on how firebase works