Android App

This README provides details about how to deploy the app, how to test it and how to add your own code and features.

Deployment

Download the APK file and run it on your device and that's it! However, make sure you have the backend, Service API and Ask Bob servers installed and running on a machine on the same network to which the app is connected to. In order to do this, clone our repository. Then go into the root of our repository and run the command 'docker-compose up' and this should automtically start up all 3 servers! Please note that the app does require a minimum of Android API level 16 (Jelly Bean 4.1.x) on the device to work.

The device running the app needs to have Google's speech recongizer installed and configured. Newer Android versions come preinstalled and configured with Google's speech recognizer. In the event that it is not installed on the device, Google's speech recognizer can be downloaded here - https://play.google.com/store/apps/details?id=com.google.android.googlequicksearchbox Once downloaded, you have to go into the device settings and search for 'Assist & voice input' (exact name may vary on Android versions but searching for 'voice' should bring up the correct setting). Then click on 'Assist & voice input' and under 'Assist app', select the Google app which you just installed. Finally, you must go into Settings, Permissions, click on the Google app which you installed and grant all the permissions relating to speech recognition.

By default, you should be able to use Google's speech recognition offline. If for some reason this is not the case, follow these instructions. Open the Google app. Select More, then Settings, then Voice, then Offline speech recognition, then ALL and add English (US).

To register/connect with the admin, ask your admin to tell you your three registration codes and simply enter these on the Register page of the app and click submit. A message should appear telling you if you have successfully connected or not.

Note: for features which utilise your location such as using the Transport service API to search for nearby train stations, location data must already exist on the device. You can ensure location data exists by going onto Google Maps and getting your current location.

Testing

The repository contains a regression test suite developed through TDD which you can use to ensure there is no functionality leakage and to which you can

add your own tests for your code. The tests under androidTest package are instrumented tests which run on a device/emulator. The tests under test package are unit tests which run on the JVM. See important notes for both below.

Instrumented tests - these tests were written using the Espresso framework and provide UI tests hence only contain tests for classes corresponding to an activity. Please note, for these tests to run, line 19 in the build.gradle file had to be commented out as these tests fail when using ANDROID_TEST_ORCHESTRATOR (nothing to do with our tests, more of an Android Studio bug).

Unit tests - these tests were written using the Robolectric framework and test the actual logic and code of the app. There are two important points to note here.

1. To run these tests, the targetSdkVersion under defaultConfig under android must be set to 29 or lower as Robolectric framework doesn't currently support any Android sdk versions greater than 29. In no way does this hinder the reliability of the tests as all the code written is intended for devices with Android sdk version 16 or higher as mentioned in the Deployment section.

2. All the tests cannot be run in one go (ie you cannot right click on the test package and select 'Run Tests in 'fisev2concierge') as for some reason this causes the tests to fail. Again, this has nothing to do with our tests, rather it is an Android Studio issue. Instead, you must run each test class individually.
   Please also ensure you have both the backend and Ask Bob servers running if you run the tests under the 'askBobConnectivity' and 'backendConnectivity' packages within the 'test' folder.

Developer Guide

The app was designed to be developer friendly, allowing developers to add their own code and features to the app. This section outlines the design of the code and sections of code itself which will helper developers.

1. Frameworks - there are two important frameworks to note. The first framework is the 'SQLiteOpenHelper' class which comes with Android. This class can be extended to allow access to the device's localised storage which is key for many of the features currently provided on the app such as Alarms. As the app's capabilities grow, it is likely that localised storage will be used more and more given the app's focus on user privacy. Our own code which uses this class can be found under 'model' package and can be used as a guide.
   The second framework, which is our own, is used for making Http requests.

Under the 'localApis' package, you can find the 'GetRequestFramework' and the 'PostRequestFramework' classes. These two classes aim to abstract away the code needed to make requests and allow developers to create instances of these classes, set the 'baseUrl' through the constructor and then make a request using the makeRequest method. The baseUrl is exactly that - the base url of the API you are connecting to. For example https://127.0.0.1/. For the GetRequestFramework class, you can then set the endpoint along with the parameters by passing them to the makeRequest method. This could be endpoint?parameter=value, for example. For the PostRequestFramework class, you can set the endpoint and the body of the request through its makeRequest method. Both classes' makeRequest methods return an ArrayList which contains the response from the API you connected to. See example code for both classes below. The idea is that if you have a private server (which does not have a public API) and you want to connect the app to it, then this can be done using the provided frameworks. An example of where this might apply is as follows. Say a hospital wishes to provide its elderly patients with the app and want to allow them to connect to their private server to retrieve information, such as which doctors in the hospital are available to them then this can be done with our frameworks. Please note that if you have a public API then you should use our plugin system to connect the app to it!

Example code:

GetRequestFramework:

GetRequestFramework getRequestFramework = new GetRequestFramework("https://127.0.0.1/");

ArrayList response = getRequestFramework.makeRequest("endpoint?parameter=value");

PostRequestFramework:

PostRequestFramework postRequestFramework = new PostRequestFramework("https://127.0.0.1/");

ArrayList response = postRequestFramework.request("endpoint", "parameter="value"");

1. UI design - the app starts up on MainActivity. As new features are added, we recognise that some of these features may require their own activity as our Alarms feature does, for example. To facilitate this, we added a scrollable view to MainActivity where you can add a button linking to your activity without having to worry about how much space is left on MainActivity.

2. Naming conventions - general good practise Android naming conventions are followed. For example, all XML files for activities are named using the activity_name convention. This ensures that those with existing Android

development experience are able to easily and quickly understand what each files/class/piece of code is supposed to do and add on their own code.

3. MVC architecture - we used this architecture to enable the Seperation of Concern principle and to make code readable and understandable. All classes which handle the Activities (views) can be found under the UI package. All classes which handle the models can be found under the 'model' package. Finally, the controllers can be found under the 'controllers' package. This allows code to be cleaner and means you can focus on developing on whichever part of the app you need to without worrying about other aspects. Furthermore, this allows for some much needed abstraction where developers can utilise existing functionality without having to worry about how it is implemeneted. For example, if a developer wishes to open a url in their code, they can just use the openUrl method in MainController without ever needing to know how this actually works. All of this has been achieved using the MVC architecture.

4. Modularised & Packaged code - the code has been split into logical packages, classes and methods to make code clean, readable and easy to understand and naviagte. This is essential for open source code to ensure anyone and everyone can easily contribute and we believe we have achieved this. With useful comments and meaningful method names as well, we hope that developers will be able to add their own code with ease.