Digital Health

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
FACULTY OF MEDICINE

Digital Health 2021/22
for
Students

Platform-Specific Code in Kotlin and Google Fit (1)

by David Kopyto

Date: November 11, 2021

Supervisors: David Kopyto, Dr. Luis Lopera, Dr. Andreas Rowald, Prof. Dr. Oliver Amft

1 Introduction

In the next two sessions we want to learn how to connect our implementation to comercial health apps such as Google Fit. Google Fit is an app which provides various health parameters. One can track activity variables like sleep. The app can be connected to various other applications, such as a smart scale or different wearables. This is done using the Google Fit API. We will use this API to read values from Google Fit and display them in our app and write values to Google Fit from our app. To achieve this, we will first need to connect from our Dart code to platfrom-specific code. In the following, we will use Kotlin code on the native side to communicate with Google Fit.

2 Google Fit Health App

Google Fit is freely available on the Play store: https://play.google.com/store/apps/details?id=com.google.android.apps.fitness&hl=en_US&gl=US. It can capture vari- ous health markers such as steps per day or sleep. Several charts can be used to examine your personal health status. We will use this app to connect our Sonar device to provide sleep information to it.

3 Kotlin MainActivity

The Google Fit's API is not written in Dart code. This means we have to migrate the data we gathered from our measurements to platform-specific code. To do this, we will first take a look at the MainActivity class in Kotlin. Please open the class under /android/app/src/main/kotlin/com.example.sonar_example_app_students. You will only see an empty MainActivity class. To correctly communicate with Dart, we will need to fill this up by instructions from the following tutorial: https://flutter.dev/docs/development/ platform-integration/platform-channels?tab=android-channel-kotlin-tab. You will find this code snippet:

```
import androidx.annotation.NonNull
import io.flutter.embedding.android.FlutterActivity import io.flutter.
embedding.engine.FlutterEngine import io.flutter.plugin.common.
MethodChannel

class MainActivity: FlutterActivity() {
    private val CHANNEL = "com.flutter.fitGate/fitGate"

    override fun configureFlutterEngine(@NonNull flutterEngine: FlutterEngine) {super . configure
        Flutter Engine ( flutter Engine ) MethodChannel(flutterEngine.dartExecutor.binary
        Messenger, CHANNEL).
            setMethodCallHandler {call ,
        result ->
        // Note : this method is invoked on the main thread .
        // TODO
    }
    }
}
```

Please note that the CHANNEL variable has been changed to another name which fits our application better. The next step is to copy this code snippet to the MainActivity class. Within the MethodCallHandler, we will later be able to send to and receive messages from the Flutter side. To do this, we first set up a function for communication on the Flutter side. This function will work with the following global variable which provides the channel for communication:

```
static const platform = const MethodChannel("com.flutter.fitGate/fitGate");
```

We will use this platform MethodChannel to communicate with the Kotlin side. This is done using the following function:

```
    void Printy () async { var data = {
          'sleeping': 'is sleeping'
        };
        String value ; try {
            value = await platform.invokeMethod('Printy', data);
              }catch (e){
                print(e);

        }
        print(value);
    }
```

The invokeMethod is key to communicate with the native side. With this you can either send or receive parameters from Kotlin. In this example, the map 'data' is passed, which just contains a simple string with a key. On the native side, this data can be extracted in the following way:

```
val args = call.argument<String>("sleeping")
```

In order to print data using the native side, please use the following function:

```
result.success(args)
```

Within the methodCallHandler, you should provide the following calls:

```
val args = call.argument<String>("sleeping") if(call.method ==
                    "Printy"){
                        result.success(args)
                        //result.success("Hello from Kotlin")

                }
```

This will provide you a very basic example of how to send and receive messages between Flutter and the native side using Kotlin. This knowledge will be useful when we set up the data channel between the Sonar app and Google Fit.

4 Google Fit API

In this section, we want to give an overview on the Google Fit API and the different functionalities it provides. First, please check the reference: https://developers.google.com/fit/android. Our final goal will be to transmit our sleep data (boolean sleep/no sleep) to the Kotlin side and convert it to an appropriate Google Fit data type. This data shall then be made accessible in Google Fit to the user. Please check the following page to take a look at the different data types of the Google Fit API: https://developers.google.com/android/reference/com/google/android/gms/fitness/FitnessActivities. These are so-called FitnessActivities. Let us now go through the workflow of integrating the API to our project step-by-step.

4.1 Authorizing your Project

To connect with Google apps, it is always necessary to authorize your project. For that, you will need to create a new project in the Google developers console: https://console.developers.google.com. Create a new project and follow the instructions. Afterwards, an OAuth Client ID needs to be created according to https://developers.google.com/fit/android/get-api-key.

4.2 Connect Fitness API to MainActivity

After permission has been granted, the requested packages from the Google Fit API can be used for our projects. Please take a look at the documentation of the Fitness API ecosys- tem first https://developers.google.com/android/reference/com/google/android/gms/fitness/package-summary. The next step will be to gather all necessary classes and packages to push sleep data from the Sonar App to Google Fit. A possible start- ing point for this can be found here: https://developers.google.com/fit/scenarios/ write-sleep-data.

5 Questions and Exercises

1. Connect your Sonar App with Kotlin. Use an async function at the Dart side and a MethodChannel at the Kotlin side. Explain your workflow. Provide a screenshot of the output at the terminal for both sending and receiving messages from and atthe Kotlin side.
2. Install Google Fit on your phone. Record three activity sessions of your choice and fill in anthropometric data like weight, height or age. What other variables could be recorded? You will find the app here: https://play.google.com/store/apps/details?id=com.google.android.apps.fitness.
3. Read through the different documentations of the Google Fit API. Explain what steps need to be taken to integrate the API to your project. The pipeline you design should help you to actually integrate the API in the next exercise session. Read specifically here https://developers.google.com/fit/android/get-started.
4. Register your project on https://console.developers.google.com/ according to https://developers.google.com/fit/android/get-api-key. Explain your steps. Please be aware that you will need a Google account to do this.
5. Look at the documentation of the Fitness API. Which packages need to be imported and which other APIs might be needed for our purposes? Explain for each package why it would be needed. Look at the different APIs here: https://developers.google.com/fit/android.
6. Develop a concept of how to migrate the data coming from the Sonar app to Google Fit. How need the data types to be converted? Are there some adjustments that have to be made in our app? If needed, feel free to make changes in the Sonar app. Explain why you make changes and document them. Use the following page as a starting point: https://developers.google.com/fit/scenarios/write-sleep-data.