My First Babble App Tutorial

Version 0.1.0

Mosaic Networks

Version 0.1.0 -- November, 2019

Table of Contents

# My First Babble App Tutorial

This article is a walkthrough of the process of building your first Android App using the babble-android library. We are working towards the Sample App included in the babble-android library[[1]](#footnote-22).

## Babble

Babble enables multiple computers to behave as one. It uses Peer to Peer (P2P) networking and a consensus algorithm to guarantee that a group of connected computers process the same commands in the same order; a technique known as state-machine replication. This makes for secure systems that can tolerate arbitrary failures, including malicious behaviour.

We use an adaptation of the Hashgraph consensus algorithm, invented by Leemon Baird. Hashgraph is best described in the [white-paper](http://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf) and its [accompanying document](http://www.swirlds.com/downloads/SWIRLDS-TR-2016-02.pdf). The original algorithm is protected by [patents](http://www.swirlds.com/ip/) in the USA. Therefore, anyone intending to use this software in the USA should obtain a license from the patent holders.

Hashgraph is based on the intuitive idea that gossiping about gossip itself yields enough information to compute a consensus ordering of events. It attains the theoretical limit of tolerating up to one-third of faulty nodes without compromising on speed. For those familiar with the jargon, it is a leaderless, asynchronous BFT consensus algorithm.

Babble projects the output of the consensus algorithm onto a linear blockchain which is more suitable for representing an ordered list of transactions and facilitates the creation of light-clients. For information about this projection please refer to [documentation](http://docs.babble.io/en/latest/blockchain.html) pages.

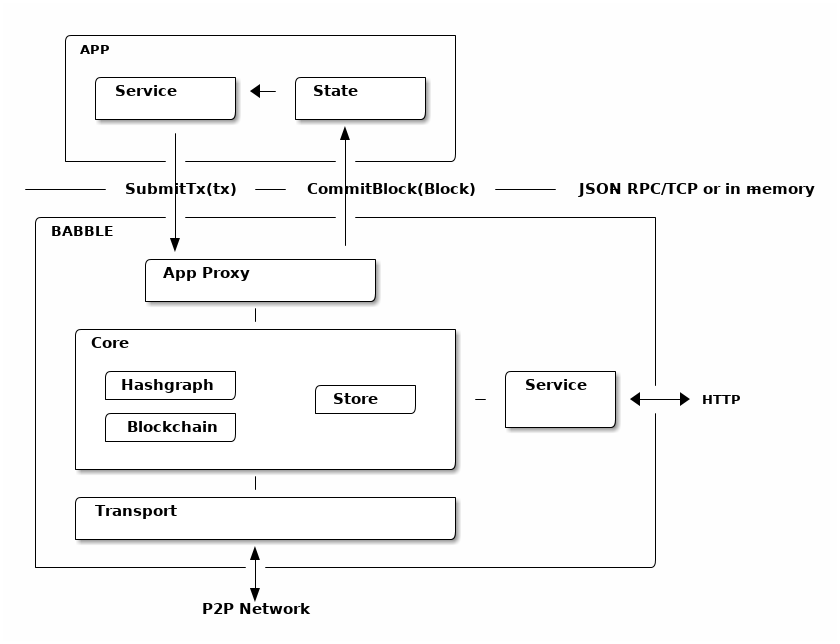
This blockchain mapping is also instrumental in two important features that were alluded to in Baird's paper, but not implemented:

* A [fast-sync](http://docs.babble.io/en/latest/fastsync.html) protocol which enables nodes to join a cluster without having to download the entire history of gossip.
* A [dynamic membership](http://docs.babble.io/en/latest/dynamic_membership.html) protocol, which enables peers to join or leave a cluster on demand.

### Design

Babble is designed to integrate with applications written in any programming language.

#### Overview



Almost any software application can be modelled in terms of a *service* and a *state*. The *service* is responsible for processing commands (ex. user input), while the *state* is responsible for manipulating and storing the data (eg. database). Usually, when commands need to update the data, the *service* will invoke the *state* directly. In a distributed application, however, commands (referred to as *transactions* in this context), must be broadcast to all replicas and consistently ordered before being applied to the *state*. This ensures that all replicas process the same commands in the same order. Hence, the *service* no longer communicates directly with the *state* (except for read-only requests), but forwards commands to a *transaction ordering system* which takes care of broadcasting and ordering the transactions across all replicas before feeding them back to the application's *state*.

Babble is an ordering system that plugs into any application thanks to a very simple interface. It uses a consensus algorithm, to replicate and order the transactions, and a blockchain to represent the resulting list. A blockchain is a linear data structure composed of batches of transactions, hashed and signed together, easily allowing to verify any transaction. So, instead of applying commands directly to the *state*, Babble applications must forward the commands to Babble and let them be processed asynchronously by the consensus system before receiving them back, in blocks, ready to be applied to the *state*.

Note that it is left to the application layer to filter out bad transactions before relaying them from clients to the consensus engine. Unlike other middleware designed to sit in front of the application (like Apache or Tendermint), the user-facing API is app-specific, and Babble just takes care or managing the consensus "under the hood". This filtering partially addresses spam from anonymous clients, but doesn't protect against malicious nodes spamming the network; that is a potential enhancement on the roadmap.

#### API

Babble communicates with the App through an AppProxy interface, which has two implementations:

* SocketProxy: A SocketProxy connects to an App via TCP sockets. It enables the application to run in a separate process or machine, and to be written in any programming language.
* InmemProxy : An InmemProxy uses native callback handlers to integrate Babble as a regular Go dependency.

The AppProxy interface exposes three methods for Babble to call the App:

* CommitBlock(Block) (CommitResponse, error): Commits a block to the application and returns the resulting state-hash, with accepted internal transactions.
* GetSnapshot(int) ([]byte, error): Gets the application snapshot corresponding to a particular block index.
* Restore([]byte) error: Restores the App state from a snapshot.

Reciprocally, AppProxy relays transactions from the App to Babble via a native Go channel - SubmitCh - which ties into the application differently depending on the type of proxy (Socket or Inmem).

Babble asynchronously processes transactions and eventually feeds them back to the App, in consensus order and bundled into blocks, with a **CommitBlock** call. Transactions are just raw bytes and Babble does not need to know what they represent. Therefore, encoding and decoding transactions is done by the App.

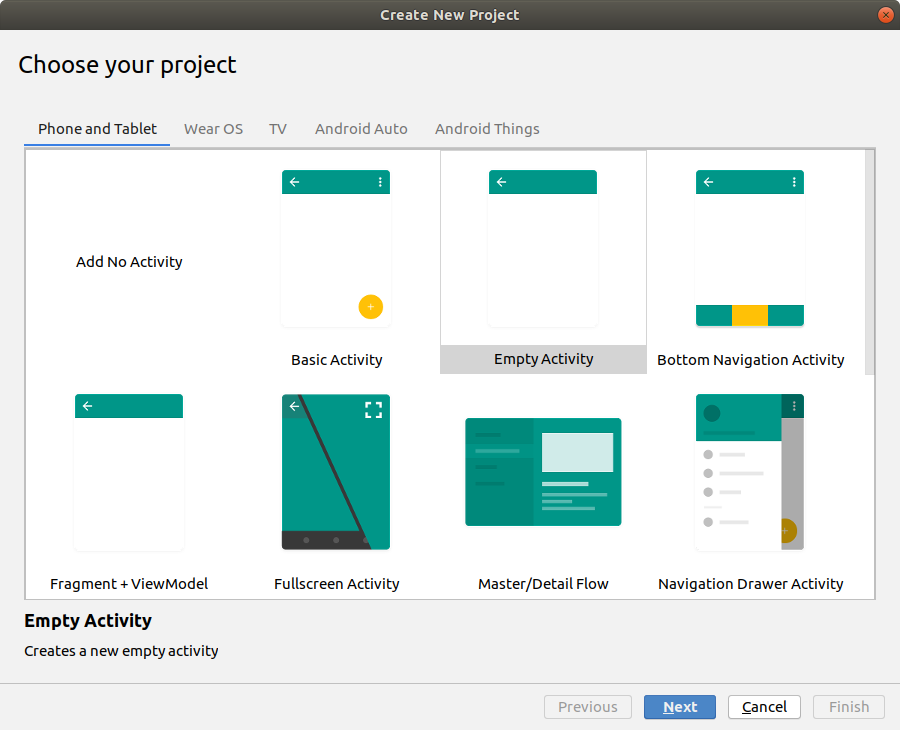
## Prerequisites

We will assume that you have installed Android Studio, an Android SDK with API version 29 and Android NDK. Android API 29 (10.0 / Q) is assumed, if use a previous version the create activity items in these instructions will use AppCompat instead of AndroidX[[2]](#footnote-36), leading to incompatibilities with the pasted source code. The babble node itself is compatible with AppCompat, but converting the sample to use AppCompat is beyond the scope of this article.

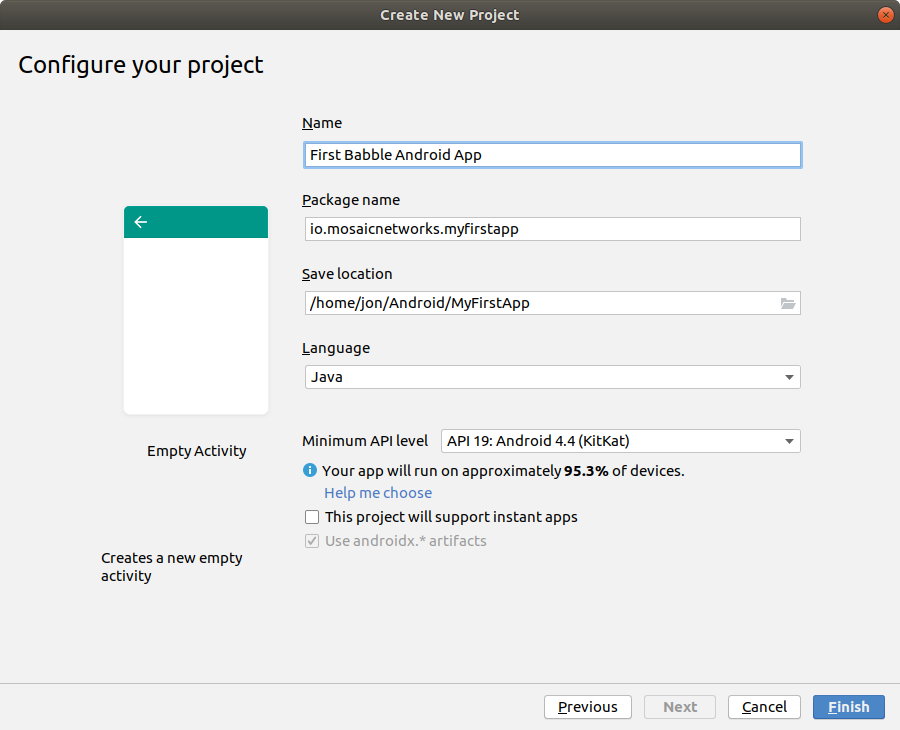
This tutorial is going to assume deployment to a physical Android device. Thus you will need an Android device (minimum API version 19[[3]](#footnote-38)) with the developer options turned on, debugging enabled, and a suitable USB cable. You could use the android emulator, but that is beyond the scope of this article.

## Our First Minimal App

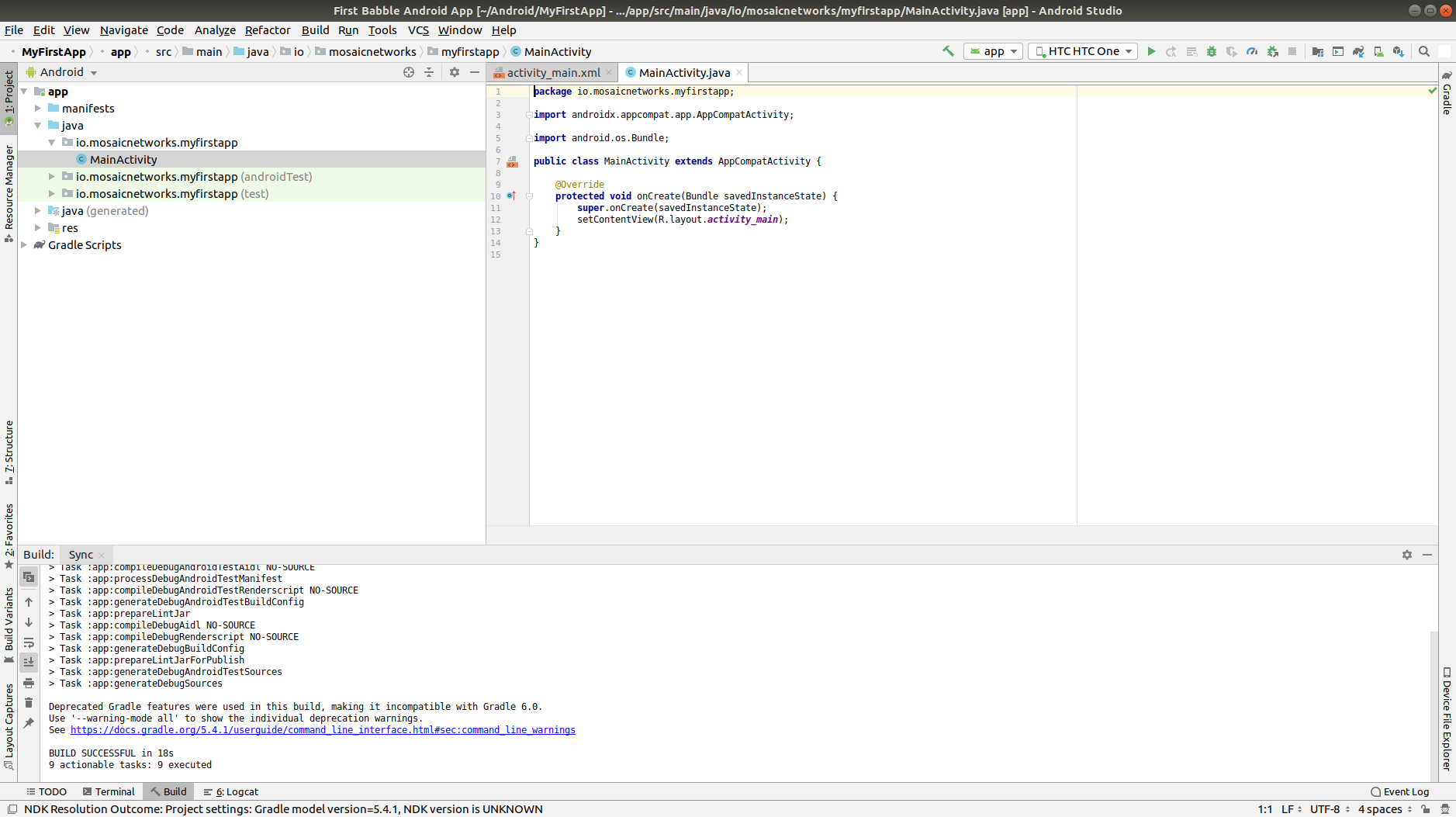
First up we will create a minimal app to test our environment and prove that we have loaded the babble-android library correctly. Fire up Android Studio and select File/New Project. You are asked to choose a project template.



Choose *Empty Activity* from the *Phone and Tablet* tab, and click Next.



The options here should be self-explanatory. We would recommend not using spaces in the Package Name or the Save Location. Do not set the Minimum API Level below 19. Click Finish.



Android Studio will open, and after expanding some of the menus it will look something like above.

*N.B. the screen will change a few seconds after opening when the initial gradle scripts complete. The status bar at the bottom of the window should tell you this is happening.*

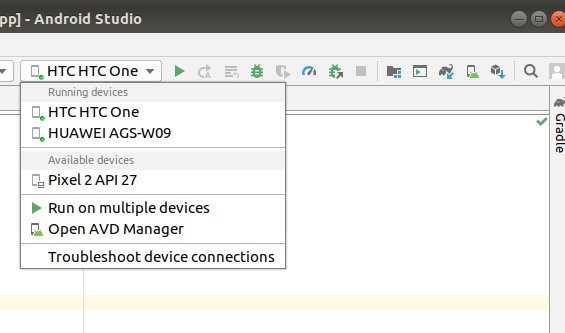
### Running the App

Connect your android device to your computer via a USB cable. First we will test that the Android Debug Bridge (**adb**) can see the device.

jon@hpjon:~/Android/MyFirstApp$ adb devices  
List of devices attached  
4JPNU18709118621 device  
jon@hpjon:~/Android/MyFirstApp$

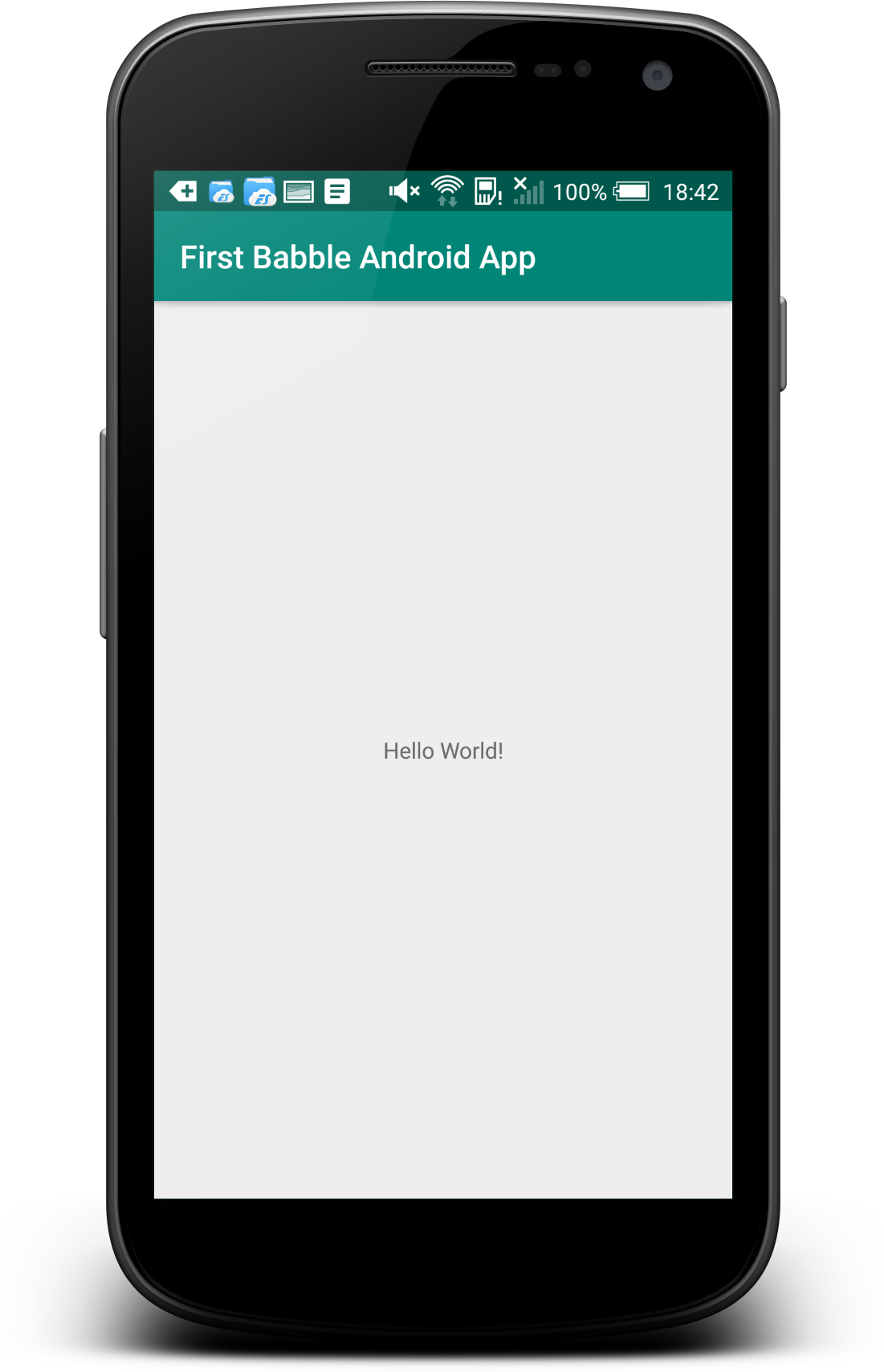
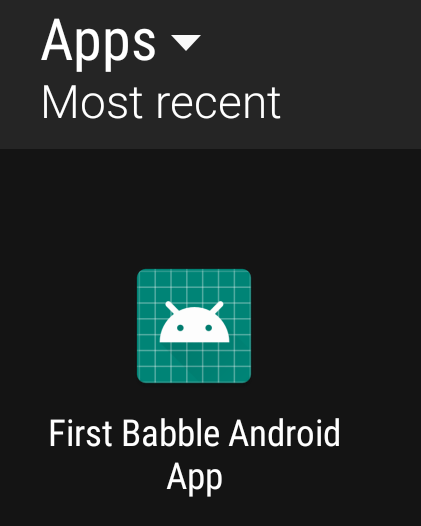
The command adb kill-server will reset this connection.

If you can see a device listed, go back to Android Studio. and in the top right is a target device dropdown. Select your device from the list.



If you cannot find your device in the list, the Troubleshoot device connections option on that menu should help.

Then press that little green triangle to the right of the dropdown device menu. Gradle then builds the app, which is then installed onto the physical android device that you selected. The whole process tool about 20 seconds on my laptop (feel free to buy me a quicker one).

If you look on the device, you should find the app installed, as on the right above.

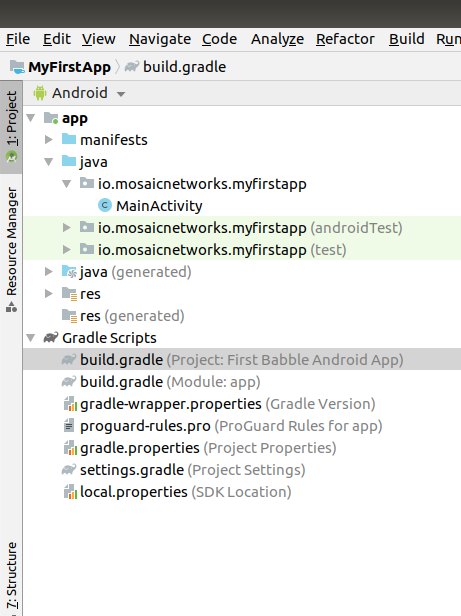
Congratulations, you have an app. Now onto Babble...

## Our First Babble App

We will now integrate the babble-android library into our skeleton app. We will use it to generate a key pair -- just to prove that we have a working library instance.

### Integrating the Babble-Android Library

The library is hosted **jcenter**. To make it available, we need to amend some gradle scripts.



In Android Studio, double click on the Project build.gradle as highlighted in the screenshot above.

We then add the 3 line maven instruction as below:

allprojects {  
 repositories {  
 google()  
 jcenter()  
 maven {  
 url 'https://dl.bintray.com/mosaicnetworks/maven'  
 }  
 }  
}

Which leaves the entire file looking like this:

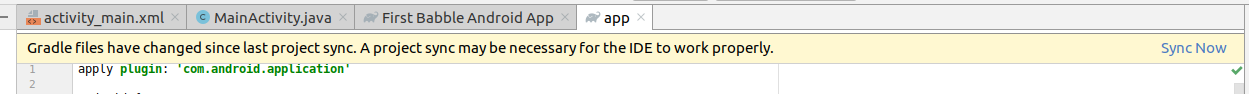
// Top-level build file where you can add configuration options common to all  
// sub-projects/modules.  
  
buildscript {  
 repositories {  
 google()  
 jcenter()  
   
 }  
 dependencies {  
 classpath 'com.android.tools.build:gradle:3.5.2'  
   
 // NOTE: Do not place your application dependencies here; they belong  
 // in the individual module build.gradle files  
 }  
}  
  
allprojects {  
 repositories {  
 google()  
 jcenter()  
 maven {  
 url 'https://dl.bintray.com/mosaicnetworks/maven'  
 }  
 }  
}  
  
task clean(type: Delete) {  
 delete rootProject.buildDir  
}

Next we need to amend the app build.gradle (it is below the Project build.gradle in the screenshot above. We add an implementation line to the bottom dependencies section.

implementation 'io.mosaicnetworks:babble:0.1.0'

This leaves us with this full file:

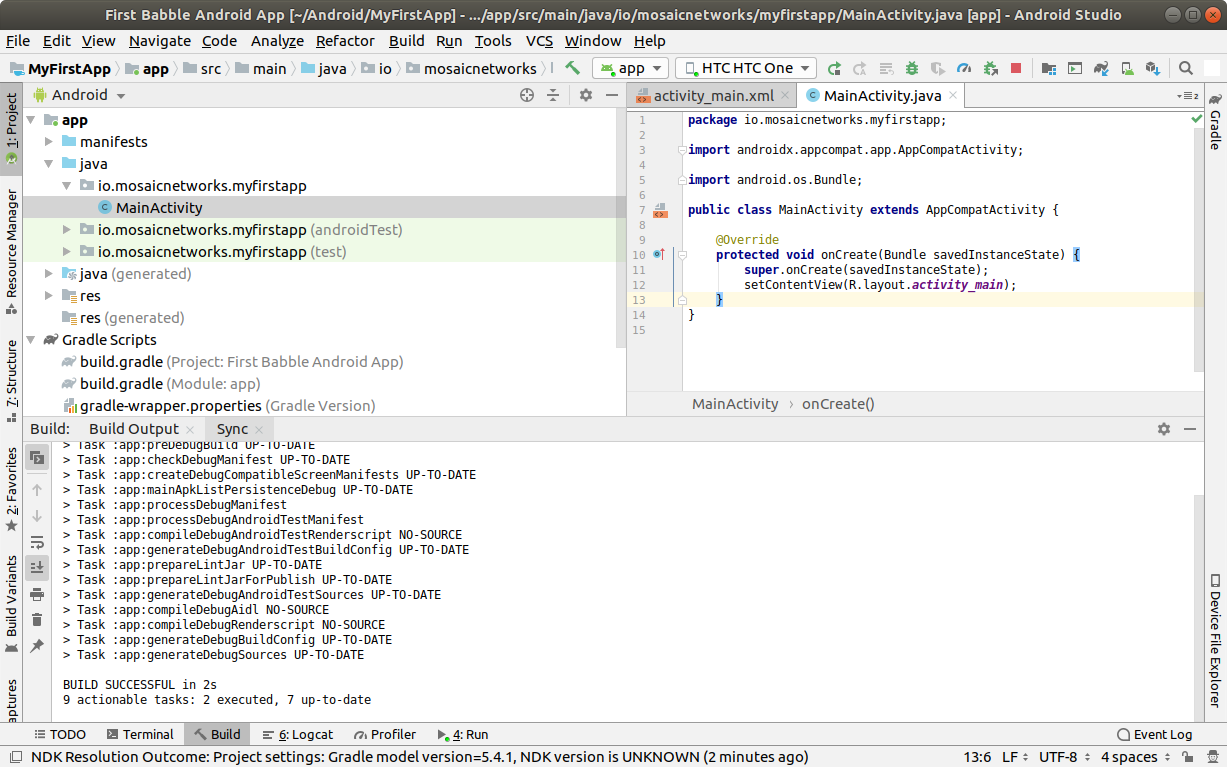
apply plugin: 'com.android.application'  
  
android {  
 compileSdkVersion 29  
 buildToolsVersion "29.0.2"  
 defaultConfig {  
 applicationId "io.mosaicnetworks.myfirstapp"  
 minSdkVersion 19  
 targetSdkVersion 29  
 versionCode 1  
 versionName "1.0"  
 testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"  
}  
buildTypes {  
 release {  
 minifyEnabled false  
 proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),  
 'proguard-rules.pro'  
 }  
 }  
}  
  
dependencies {  
 implementation fileTree(dir: 'libs', include: ['\*.jar'])  
 implementation 'io.mosaicnetworks:babble:0.2.1'  
 implementation 'androidx.appcompat:appcompat:1.1.0'  
 implementation 'androidx.constraintlayout:constraintlayout:1.1.3'  
 testImplementation 'junit:junit:4.12'  
 androidTestImplementation 'androidx.test.ext:junit:1.1.1'  
 androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'  
}



When you get a message like the above, click the Sync Now link on the right of the message.

### Using the Library

The library should now be included in the project. So lets use it! Open MainActivity.java as below:



Add the lines below underneath the last import statement. The lines will appear greyed out, as the import is not yet used. As well as babble we are importing the Log package to write to the Android logs.

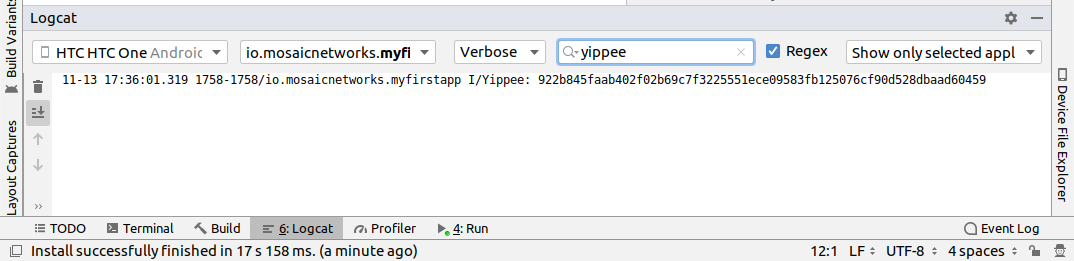
import io.mosaicnetworks.babble.node.KeyPair;  
import android.util.Log;

Add the following lines as the last line of the onCreate function

KeyPair kp = new KeyPair();  
Log.i("Yippee",kp.privateKey);

This code generates a key pair and writes the private code to the logs.

Save all the files and run your app.



The app looks exactly as per the previous iteration, so lets take a look under the hood. Press logcat, as highlighted in gray in the screenshot above. Then type yippee in the search box at the top of that window to filter the logs. You should have a freshly generated private key in there.

This project at this stage is available from github from [here](https://github.com/mosaicnetworks/babble-android-tutorial/tree/stage1) [[4]](#footnote-55)

The downloadable version of the project has mosaic network icons, rather than the default android ones. You can customise the icons using [Android Studio Image Asset Studio](https://developer.android.com/studio/write/image-asset-studio).[[5]](#footnote-57)

## Our First Babble Blockchain

Now we have access to the babble library from within our app, the next stage is to start a babble network. We will start with a single node. But before we can start babble we need to add some UI elements to allow us to interact with our babble node.

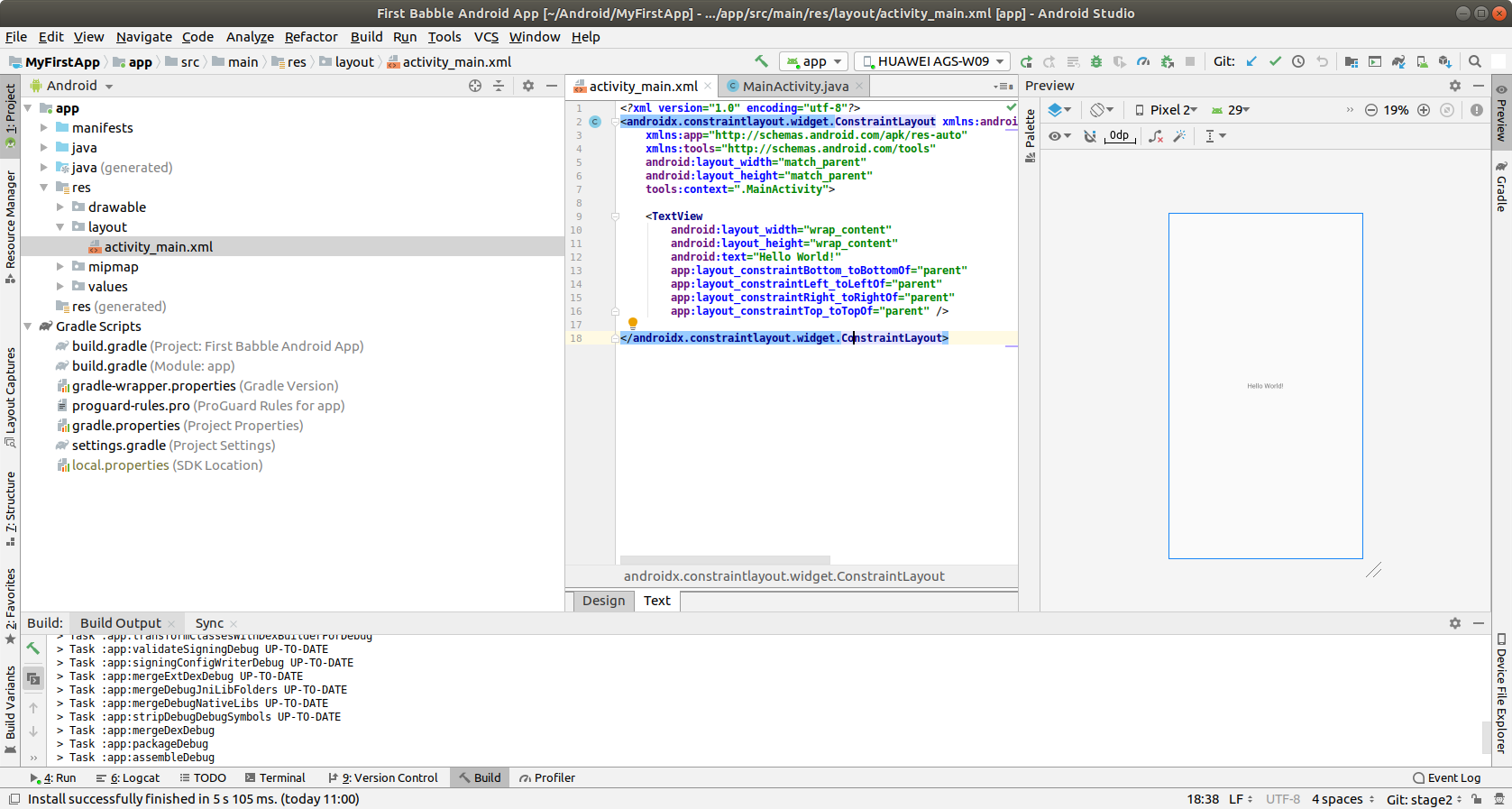
Currently our application launches the activity MainActivity which calls the key pair generation code in it's onCreate method.

In the Sample App [[6]](#footnote-59) that we are working towards, the MainActivity Screen presents the user with a choice of "**New**" or "**Join**". **New** starts a new babble network with your device as the sole peer. **Join** lets you specify the address of an existing network, pull down the configuration for that network and request to join it.

As **New** is standalone functionality, and **Join** requires **New** be implemented to function, we will implement **New** first.

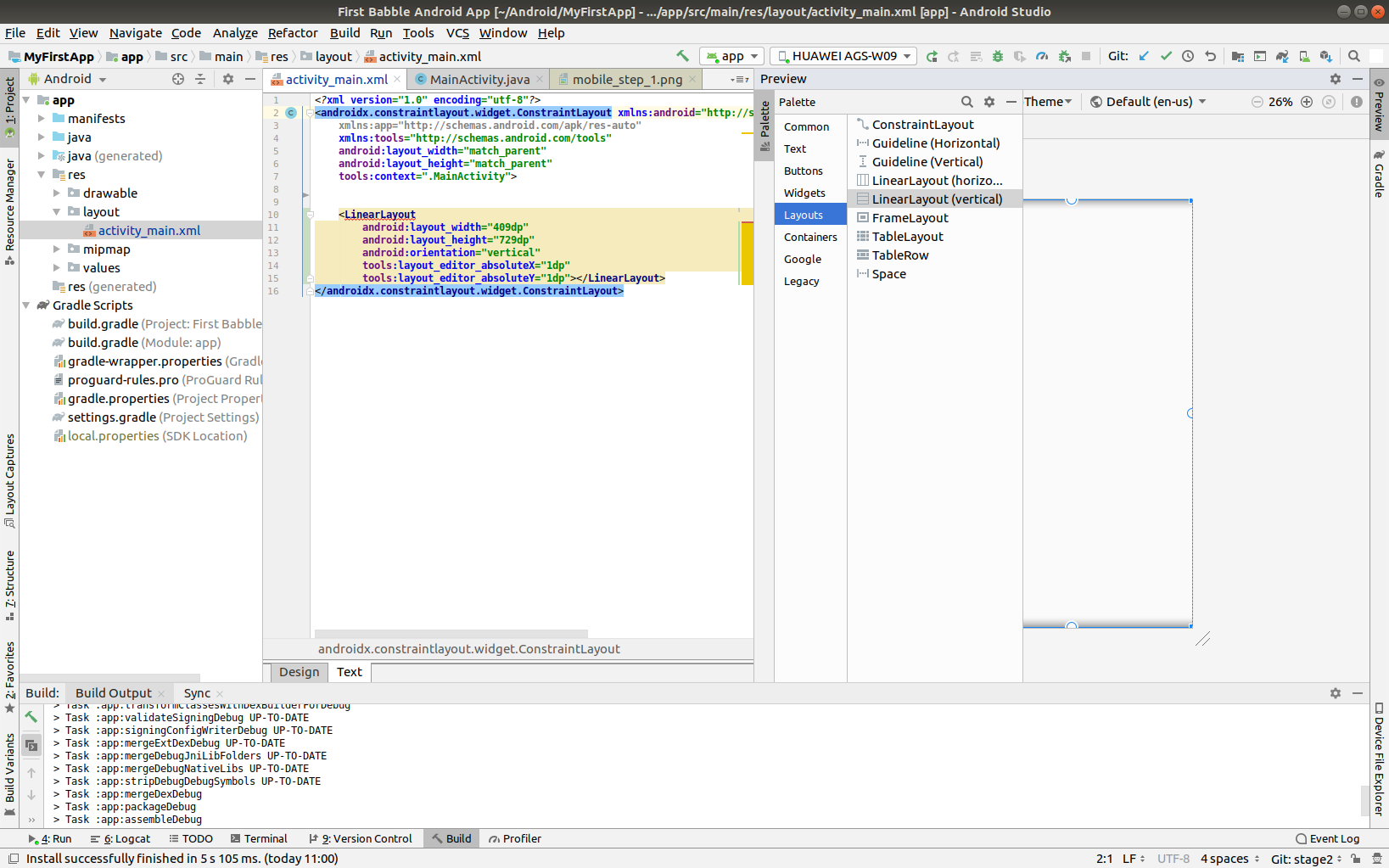
### Main Activity

First up we will amend MainActivity.java. In the Android view it's layout should be under app/res/layout/activity\_main.xml. Double click on that file to open the XML layout file. It may also show a Preview:



Delete the TextView tag. We have passed beyond the "Hello World!" stage.

In the preview window. Click on Palette / Layouts / LinearLayout (vertical) and drag it onto the preview window.



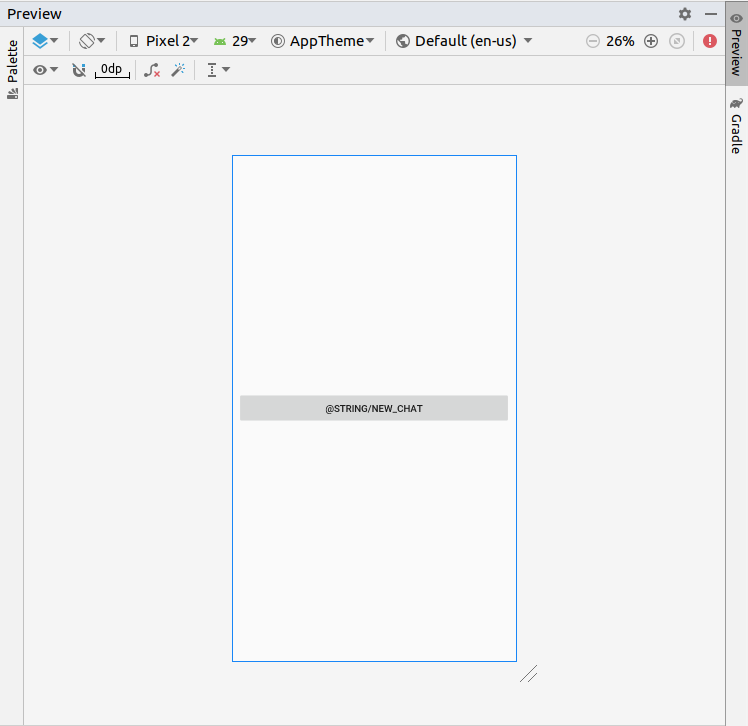
We will then set some properties for the LinearLayout. Amend it to look like this:

<LinearLayout  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginStart="8dp"  
 android:layout\_marginLeft="8dp"  
 android:layout\_marginTop="8dp"  
 android:layout\_marginEnd="8dp"  
 android:layout\_marginRight="8dp"  
 android:layout\_marginBottom="8dp"  
 android:orientation="vertical"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent">  
   
 </LinearLayout>

The principal effect of theses changes are to set a margin so the buttons will not reach to the edge of the screen. Next we add a button by adding the following code between the opening and closing LinearLayout tags.

<Button  
 android:id="@+id/button3"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:onClick="newChat"  
 android:text="@string/new\_chat" />

Note that the onClick method is marked as an error as we have not created it yet, as is the caption. The preview should look as below:



Update the strings.xml file found under app/res/values/strings.xml adding the line:

<string name="new\_chat">New</string>

to give:

<resources>  
 <string name="app\_name">First Babble Android App</string>  
 <string name="new\_chat">New</string>  
</resources>

Save the file, and if you navigate back to the MainActivity preview, the text New should now appear on the button.

Open MainActivity.java (the java code, not the XML).

Add the following code to the MainActivity class, under the onCreate method:

// called when the user presses the new chat button  
 public void newChat(View view) {  
 Log.i("Ki","newChat Called");  
 }

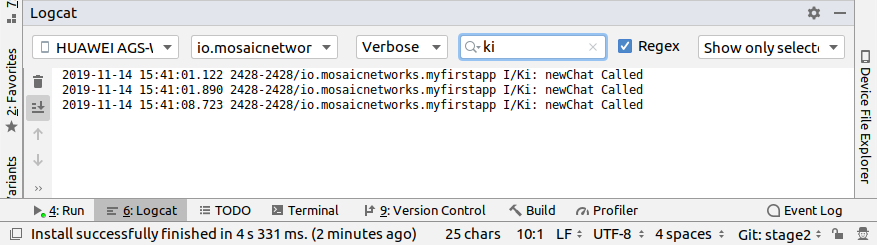
Add the line below as the first line of the class, we will use this later to identified log messages from our app. :

public class MainActivity extends AppCompatActivity {  
  
 public static final String TAG = "FIRST-BABBLE-APP";

If the IDE does not add the line below, add it manually:

import android.view.View;

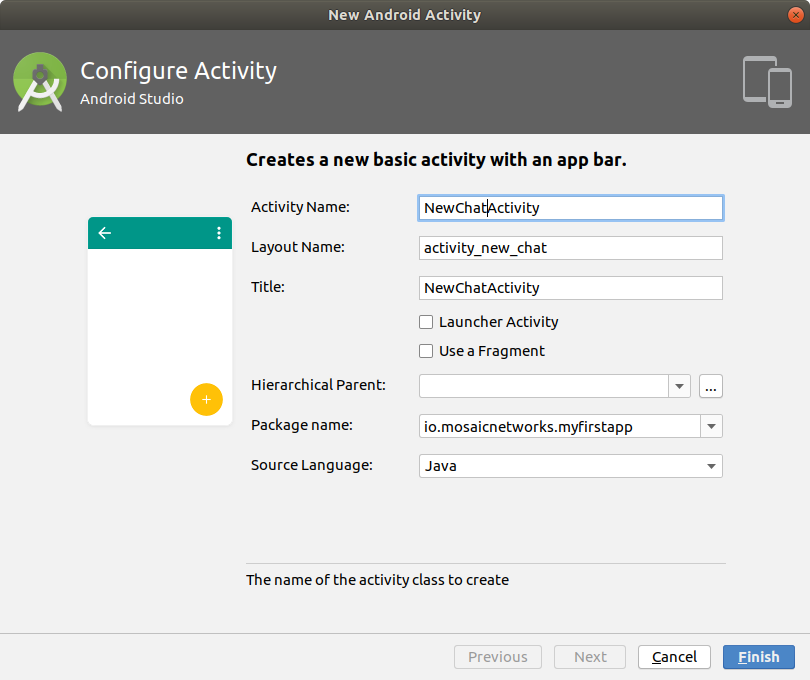
The project should now build successfully. So build and run it. If you search for Ki in the logcat window --- exactly as you searched for Yippee before you should find results like below (if you have pressed the New button a few times...).



## New Chat Activity

When we press the New button, we want to open a new screen where we can set some Chat Options. So lets create NewChatActivity.

Right click on MainActivity under the java tree. Select New > Activity > Basic Activity and enter the details as below:



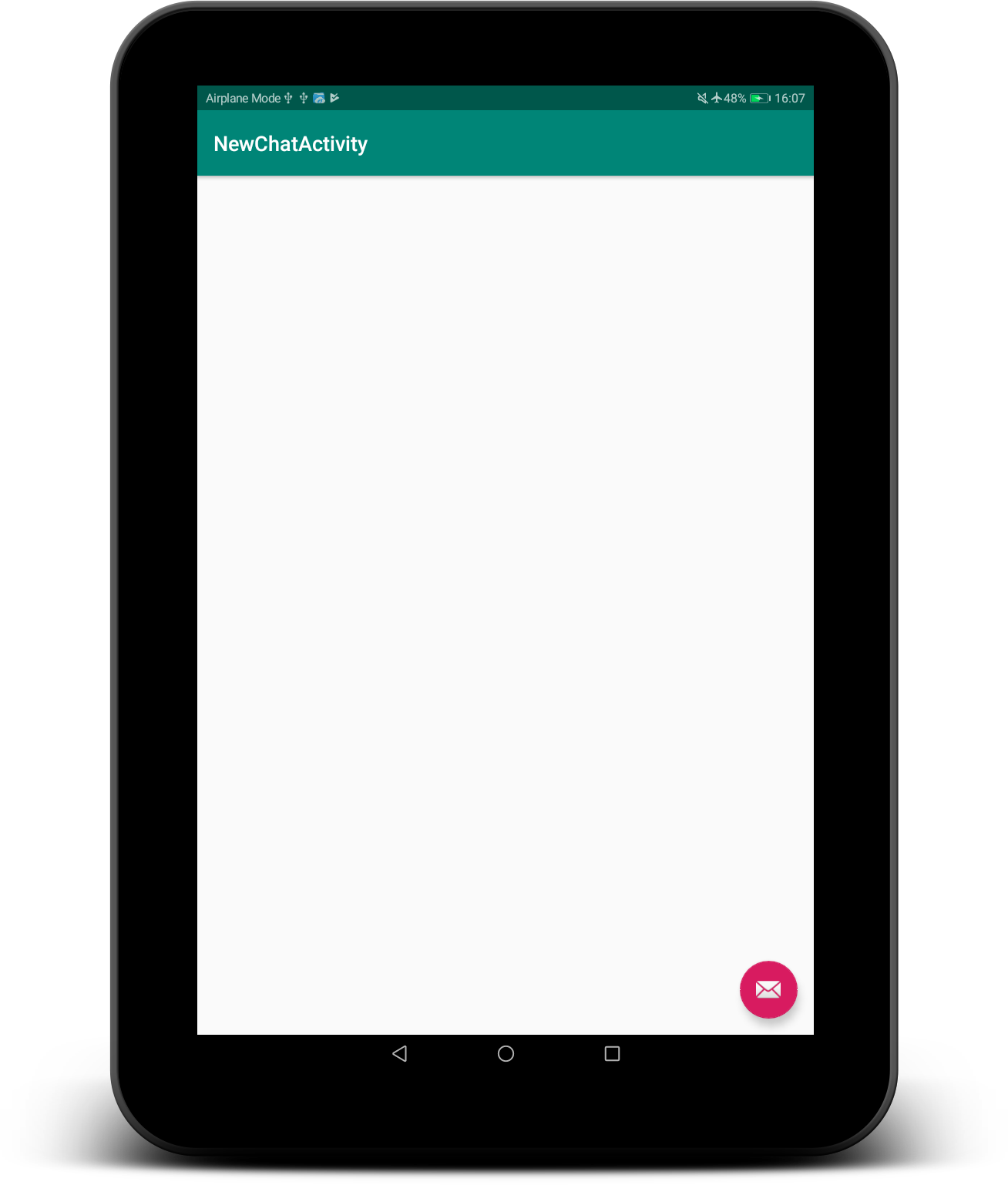
Edit MainActivity.java and amend the newChat function to read as follows:

// called when the user presses the new chat button  
 public void newChat(View view) {  
 Log.i("Ki","newChat Called");  
 Intent intent = new Intent(this, NewChatActivity.class);  
 startActivity(intent);  
 }

If the IDE does not add the line below, add it manually:

import android.content.Intent;

This change means pressing the button opens the NewChat Activity. The app will compile and run at this point. After pressing the button, you should get a screen that looks like this:

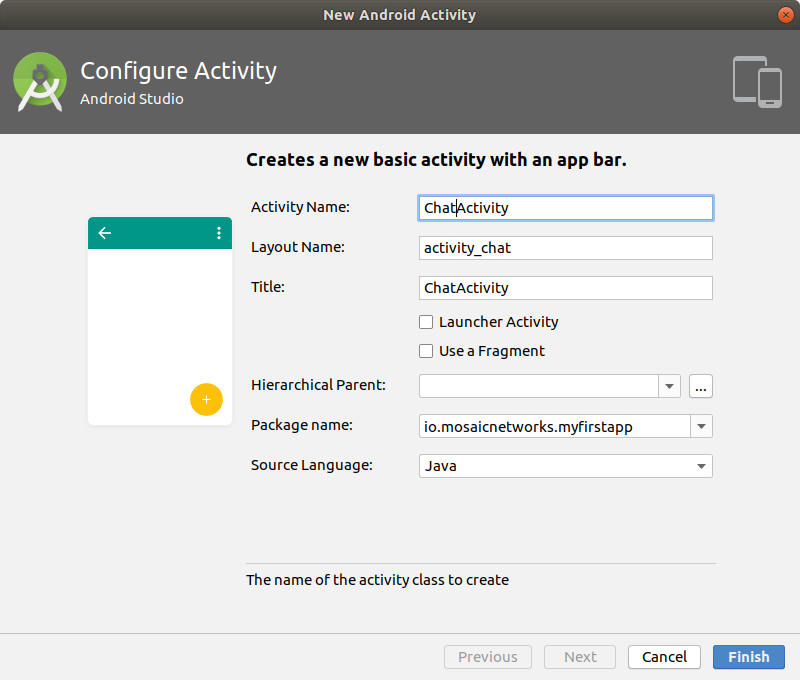


Update res/values/strings.xml to set the title\_activity\_new\_chat to "New Chat"

<resources>  
 <string name="app\_name">First Babble Android App</string>  
 <string name="new\_chat">New</string>  
 <string name="title\_activity\_new\_chat">New Chat</string>  
</resources>

### New Chat Activity Layout

Create new BasicActivity activity ChatActivity as per the NewChatActivity abov, using the values below:



In activity\_new\_chat.xml replace the whole file with the following code:

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout   
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".NewChatActivity">  
  
 <LinearLayout  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginStart="8dp"  
 android:layout\_marginLeft="8dp"  
 android:layout\_marginTop="8dp"  
 android:layout\_marginEnd="8dp"  
 android:layout\_marginRight="8dp"  
 android:layout\_marginBottom="8dp"  
 android:orientation="vertical"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent">  
  
 <EditText  
 android:id="@+id/editText"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:ems="10"  
 android:hint="@string/moniker"  
 android:inputType="textPersonName" />  
  
 <Button  
 android:id="@+id/button"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:onClick="startChat"  
 android:text="@string/start" />  
</LinearLayout>  
   
</androidx.constraintlayout.widget.ConstraintLayout>

This adds and edit box to set the moniker for this node and a button to actually start the Chat.

You can delete content\_new\_chat.xml from the res/layout folder now as we have just removed the reference to it.

In NewChatActivity.java replace the whole file with the code below:

package io.mosaicnetworks.myfirstapp;  
  
import android.content.Intent;  
import android.os.Bundle;  
import androidx.annotation.StringRes;  
import androidx.appcompat.app.AlertDialog;  
import androidx.appcompat.app.AppCompatActivity;  
import android.view.View;  
import android.widget.EditText;  
import java.util.ArrayList;  
import io.mosaicnetworks.babble.discovery.Peer;  
  
  
public class NewChatActivity extends AppCompatActivity {  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_new\_chat);  
 }  
  
 // called when the user presses the start chat button  
 public void startChat(View view) {  
 //get moniker  
 EditText editText = findViewById(R.id.editText);  
 String moniker = editText.getText().toString();  
 if (moniker.isEmpty()) {  
 displayOkAlertDialog(R.string.no\_moniker\_alert\_title,   
 R.string.no\_moniker\_alert\_message);  
 return;  
 }  
  
 MessagingService messagingService = MessagingService.getInstance();  
 try {  
 messagingService.configureNew(moniker, Utils.getIPAddr(this));  
 } catch (IllegalStateException ex) {  
 //we tried to reconfigure before a leave completed  
 displayOkAlertDialog(R.string.babble\_busy\_title,   
 R.string.babble\_busy\_message);  
 return;  
 }  
  
 messagingService.start();  
 Intent intent = new Intent(this, ChatActivity.class);  
 intent.putExtra("MONIKER", moniker);  
 startActivity(intent);  
 }  
  
 private void displayOkAlertDialog(@StringRes int titleId,   
 @StringRes int messageId) {  
 AlertDialog alertDialog = new AlertDialog.Builder(this)  
 .setTitle(titleId)  
 .setMessage(messageId)  
 .setNeutralButton(R.string.ok\_button, null)  
 .create();  
 alertDialog.show();  
 }  
}

### AppState.java

Copy the source below into place in the same folder as MainActivity.java:

package io.mosaicnetworks.myfirstapp;  
  
import com.google.gson.JsonSyntaxException;  
  
import java.nio.charset.Charset;  
import java.nio.charset.StandardCharsets;  
import java.security.MessageDigest;  
import java.security.NoSuchAlgorithmException;  
  
import io.mosaicnetworks.babble.node.BabbleState;  
  
public class AppState implements BabbleState {  
  
 private Message mLatestMessage;  
 private static final MessageDigest mSha256Digest;  
 private byte[] mStateHash = "genesis-state".getBytes();  
  
 static {  
 try {  
 mSha256Digest = MessageDigest.getInstance("SHA-256");  
 } catch (NoSuchAlgorithmException ex) {  
 // Every implementation of the Java platform is required  
 // to support the SHA-256 MessageDigest algorithm,   
 // so we shouldn't get here!  
 throw new RuntimeException(ex);  
 }  
 }  
  
 @Override  
 public byte[] applyTransactions(byte[][] transactions) {  
 for (byte[] rawTx:transactions) {  
 String tx = new String(rawTx, StandardCharsets.UTF\_8);  
  
 BabbleTx babbleTx;  
 try {  
 babbleTx = BabbleTx.fromJson(tx);  
 } catch (JsonSyntaxException ex) {  
 //skip any malformed transactions  
 continue;  
 }  
  
 onMessageReceived(Message.fromBabbleTx(babbleTx));  
  
 }  
  
 return new byte[0];  
 }  
  
 @Override  
 public void reset() {  
 //do nothing  
 }  
  
 private void onMessageReceived(Message message) {  
  
 mLatestMessage = message;  
 }  
  
 public Message getLatestMessage() {  
 //TODO: this can return null if no messages are successfully parsed  
 return mLatestMessage;  
 }  
  
 //TODO: use state hash  
 private void updateStateHash(String tx) {  
 mStateHash = hashFromTwoHashes(mStateHash, hash(tx));  
 }  
  
 private static byte[] hash(String tx) {  
 return mSha256Digest.digest(tx.getBytes(Charset.forName("UTF-8")));  
 }  
  
 private static byte[] hashFromTwoHashes(byte[] a, byte[] b) {  
 byte[] tempHash = new byte[a.length + b.length];  
 System.arraycopy(a, 0, tempHash, 0, a.length);  
 System.arraycopy(b, 0, tempHash, 0, b.length);  
 return mSha256Digest.digest(tempHash);  
 }  
}

### BabbleTx.java

Copy the source below into place in the same folder as MainActivity.java:

package io.mosaicnetworks.myfirstapp;  
  
import com.google.gson.Gson;  
import com.google.gson.annotations.SerializedName;  
  
public class BabbleTx implements io.mosaicnetworks.babble.node.BabbleTx {  
  
 private final static Gson gson = new Gson();  
  
 @SerializedName("from")  
 public final String from;  
  
 @SerializedName("text")  
 public final String text;  
  
 public BabbleTx(String from, String text) {  
 this.from = from;  
 this.text = text;  
 }  
  
 public static BabbleTx fromJson(String txJson) {  
 return gson.fromJson(txJson, BabbleTx.class);  
 }  
  
 public byte[] toBytes() {  
 return gson.toJson(this).getBytes();  
 }  
}

### Message.java

Copy the source below into place in the same folder as MainActivity.java:

package io.mosaicnetworks.myfirstapp;  
  
import com.google.gson.Gson;  
import com.stfalcon.chatkit.commons.models.IMessage;  
import com.stfalcon.chatkit.commons.models.IUser;  
  
import java.util.Date;  
  
public final class Message implements IMessage {  
  
 private final static Gson gson = new Gson();  
  
 public final static class Author implements IUser {  
  
 private final String mName;  
  
 public Author(String name) {  
 mName = name;  
 }  
  
 @Override  
 public String getId() {  
 return mName;  
 }  
  
 @Override  
 public String getName() {  
 return mName;  
 }  
  
 @Override  
 public String getAvatar() {  
 return null;  
 }  
 }  
  
 private final String mText;  
 private final String mAuthor;  
 private final Date mDate;  
  
 public Message(String text, String author) {  
 mText = text;  
 mAuthor = author;  
 mDate = new Date();  
 }  
  
 public static Message fromBabbleTx(BabbleTx babbleTx) {  
 return new Message(babbleTx.text, babbleTx.from);  
 }  
  
 public BabbleTx toBabbleTx() {  
 return new BabbleTx(mAuthor, mText);  
 }  
  
 @Override  
 public String getId() {  
 return mAuthor;  
 }  
  
 @Override  
 public String getText() {  
 return mText;  
 }  
  
 @Override  
 public Author getUser() {  
 return new Author(mAuthor);  
 }  
  
 @Override  
 public Date getCreatedAt() {  
 return mDate;  
 }  
  
}

You will note the section below introduces an external dependency:

import com.stfalcon.chatkit.commons.models.IMessage;  
import com.stfalcon.chatkit.commons.models.IUser;

Add the lines below to the app build.gradle file dependencies section, and click Sync Now on the pop up bar:

implementation 'com.google.code.gson:gson:2.8.5'  
 implementation 'com.github.stfalcon:chatkit:0.3.3'

### Utils.java

Copy the source below into place in the same folder as MainActivity.java:

package io.mosaicnetworks.myfirstapp;  
  
import android.content.Context;  
import android.net.wifi.WifiManager;  
import android.text.format.Formatter;  
import android.util.Log;  
  
import static android.content.Context.WIFI\_SERVICE;  
  
public class Utils {  
  
 public static String getIPAddr(Context context) {  
 WifiManager wm = (WifiManager)   
 context.getApplicationContext().getSystemService(WIFI\_SERVICE);  
 String ip =   
 Formatter.formatIpAddress(wm.getConnectionInfo().getIpAddress());  
 Log.d("getIPAddr", "Got IP address: " + ip);  
 return ip;  
 }  
}

### MessagingService.java

Copy the source below into place in the same folder as MainActivity.java:

package io.mosaicnetworks.myfirstapp;  
  
import io.mosaicnetworks.babble.node.BabbleService;  
  
public final class MessagingService extends BabbleService<AppState> {  
  
 private static MessagingService INSTANCE;  
  
 public static MessagingService getInstance() {  
 if (INSTANCE==null) {  
 INSTANCE = new MessagingService();  
 }  
  
 return INSTANCE;  
 }  
  
 private MessagingService() {  
 super(new AppState());  
 }  
}

### strings.xml

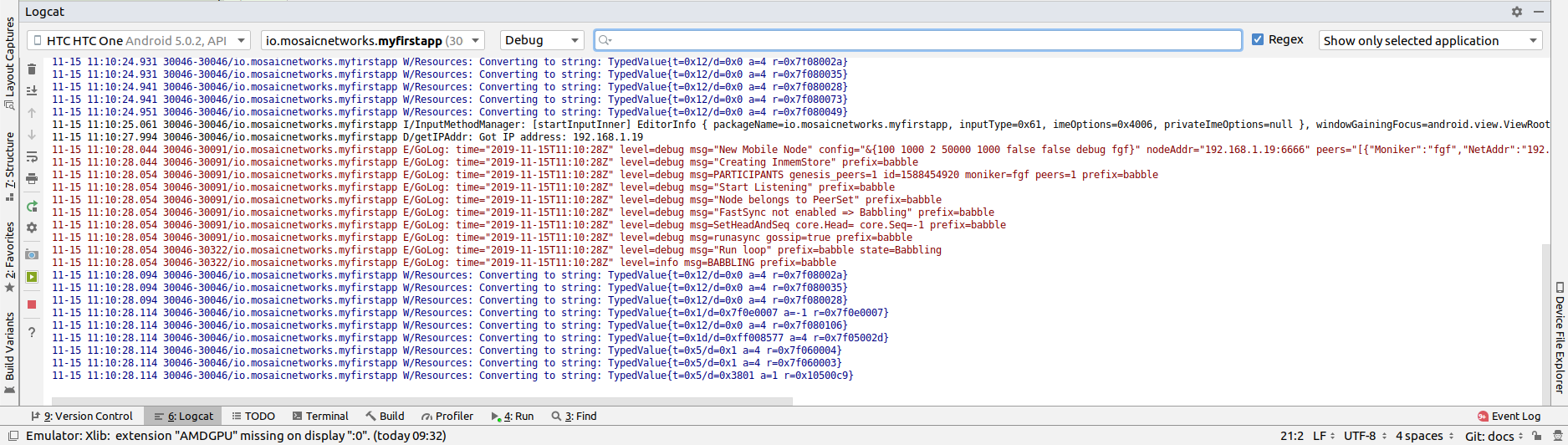
We need to the add the following to res/values/strings.xml as they are used in the code changes above.

<string name="moniker">Moniker</string>  
 <string name="start">Start</string>  
 <string name="no\_moniker\_alert\_title">No moniker</string>  
 <string name="no\_moniker\_alert\_message">Please enter a moniker!</string>  
 <string name="ok\_button">OK</string>  
 <string name="babble\_busy\_title">Babble node busy</string>  
 <string name="babble\_busy\_message">Please try again in a few seconds!</string>

### Running Babble

And finally after all of that cut and paste, we have a working instance of babble --- albeit with at least one major drawback --- it has no UI and no way to access it.

If you start the app through Android Studio, and look at the logcat output (filtered to just our app), after pressing the New button, entering a Moniker and pressing the Join button, you should see something like below:



## Some Explanations

We have just added a lot of code, which is all co-dependent. Now we have a babble invocation in place, we can pause to explain what just happened there.

Within the NewChatActivity.java button click handler the contents of the edit box are processed into mMoniker and the function joinChat() is called.

Within joinChat() are the 2 key lines:

messagingService.configureNew(moniker, Utils.getIPAddr(this));  
  
 ...  
  
 messagingService.start();

They invoke the singleton instance of the MessagingService class. The MessagingService class is a simplified version of the code from the sample app included in the babble-android repo.

At a high level, it configures, then starts a babble node.

Delving a little deeper, the configureNew method is a wrapper for a configure method that takes a peers list, the moniker and IP for this node. In our New Node use case, the peers list will always be empty.

This project at this stage is available from github from [here](https://github.com/mosaicnetworks/babble-android-tutorial/tree/stage2) [[7]](#footnote-80)

## Interacting with Babble

The next stage is to make Babble usable. To do that we need to work on the ChatActivity so it sends and receives messages from Babble.

First up we need a UI. We are going to use [ChatKit](https://github.com/stfalcon-studio/ChatKit) rather than reinvent the wheel.

### build.gradle (app)

We need to add this library to the app build.gradle file by adding the following line:

implementation 'com.github.stfalcon:chatkit:0.3.3'

### activity\_chat.xml

We can then add the layout to res/layout/activity\_chat.xml --- replace all the contents with the code below:

<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:background="@color/white"  
 tools:context=".ChatActivity">  
  
 <com.stfalcon.chatkit.messages.MessagesList  
 android:id="@+id/messagesList"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:layout\_above="@+id/input"/>  
  
 <View  
 android:layout\_width="match\_parent"  
 android:layout\_height="1dp"  
 android:layout\_above="@+id/input"  
 android:layout\_marginLeft="16dp"  
 android:layout\_marginRight="16dp"  
 android:background="@color/gray\_light"/>  
  
 <com.stfalcon.chatkit.messages.MessageInput  
 android:id="@+id/input"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_alignParentBottom="true"  
 app:inputHint="@string/hint\_enter\_a\_message"  
 app:showAttachmentButton="true"/>  
  
</RelativeLayout>

The layout here is a fairly standard chat layout a message entry section at the bottom of the screen and a message display above it.

### colors.xml

We need to the add the following to res/values/colors.xml as it is used in the code changes above.

<color name="gray\_light">#e8e8e8</color>

### ChatActivity.java

Replace all of the file ChatActivity.java with the code below:

package io.mosaicnetworks.myfirstapp;  
  
import android.os.Bundle;  
import androidx.appcompat.app.AppCompatActivity;  
import android.content.Intent;  
import android.widget.Toast;  
import com.stfalcon.chatkit.messages.MessageInput;  
import com.stfalcon.chatkit.messages.MessagesList;  
import com.stfalcon.chatkit.messages.MessagesListAdapter;  
  
import io.mosaicnetworks.babble.node.ServiceObserver;  
  
public class ChatActivity extends AppCompatActivity implements ServiceObserver {  
  
  
 private MessagesListAdapter<Message> mAdapter;  
 private String mMoniker;  
 private final MessagingService mMessagingService =  
 MessagingService.getInstance();  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_chat);  
  
 Intent intent = getIntent();  
 mMoniker = intent.getStringExtra("MONIKER");  
  
 initialiseAdapter();  
 mMessagingService.registerObserver(this);  
  
 if (mMessagingService.getState()!=  
 MessagingService.State.RUNNING\_WITH\_DISCOVERY) {  
 Toast.makeText(this, "Unable to advertise peers",  
 Toast.LENGTH\_LONG).show();  
 }  
 }  
  
 private void initialiseAdapter() {  
 MessagesList mMessagesList = findViewById(R.id.messagesList);  
  
 mAdapter = new MessagesListAdapter<>(mMoniker, null);  
 mMessagesList.setAdapter(mAdapter);  
  
 MessageInput input = findViewById(R.id.input);  
  
 input.setInputListener(new MessageInput.InputListener() {  
 @Override  
 public boolean onSubmit(CharSequence input) {  
 mMessagingService.submitTx(new Message(input.toString(),  
 mMoniker).toBabbleTx());  
 return true;  
 }  
 });  
 }  
  
 @Override  
 public void stateUpdated() {  
 final Message message = mMessagingService.state.getLatestMessage();  
  
 runOnUiThread(new Runnable() {  
 @Override  
 public void run() {  
 mAdapter.addToStart(message, true);  
 }  
 });  
 }  
  
 @Override  
 public void onBackPressed() {  
 mMessagingService.leave(null);  
 super.onBackPressed();  
 }  
  
 @Override  
 protected void onDestroy() {  
 mMessagingService.removeObserver(this);  
  
 super.onDestroy();  
 }  
}

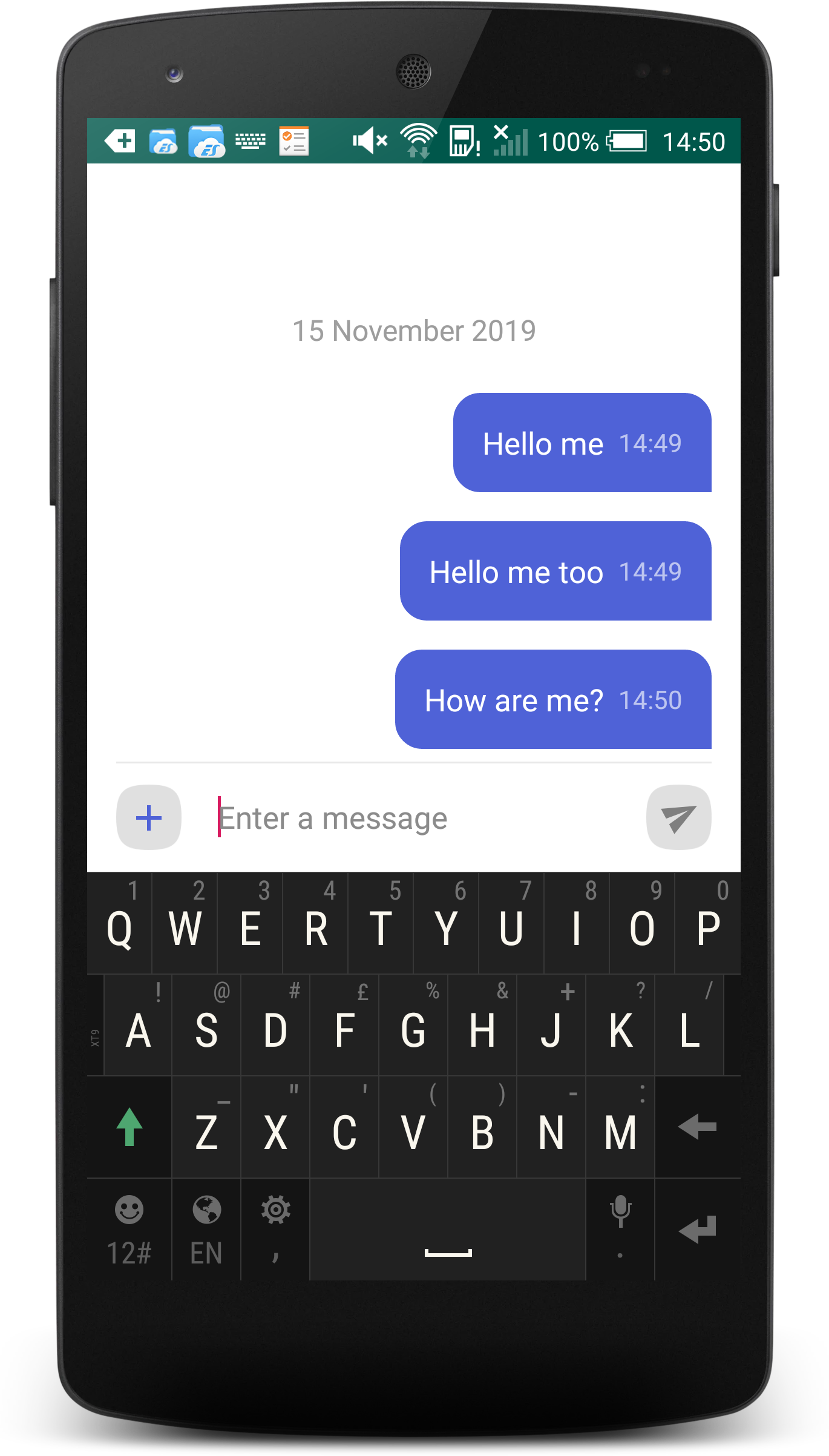
### strings.xml

We need to the add the following to res/values/strings.xml as they are used in the code changes above.

<string name="hint\_enter\_a\_message">Enter a message</string>

### Build, Run and Test

Build your app and run it. You should now be able to start a chat with yourself and send messages to yourself as below:



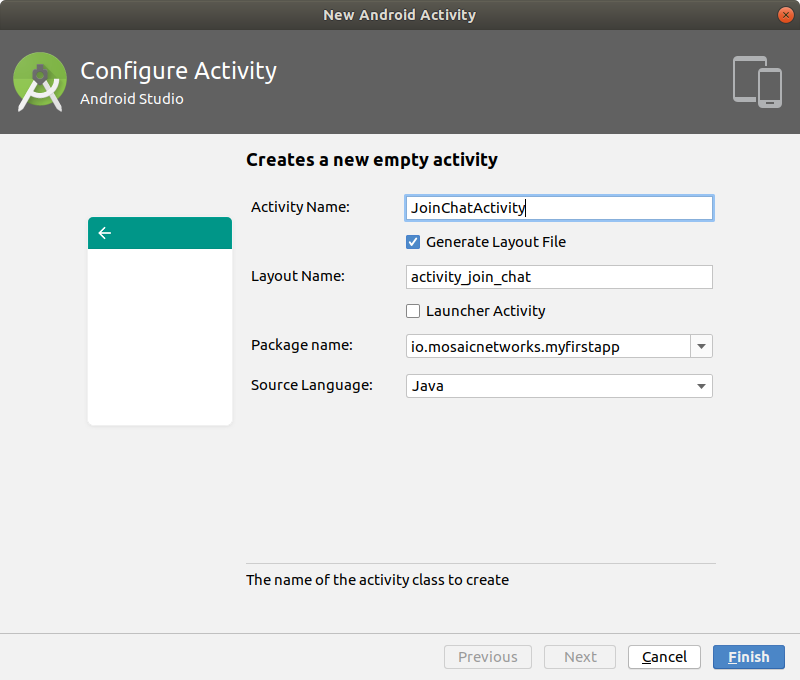
This project at this stage is available from github from [here](https://github.com/mosaicnetworks/babble-android-tutorial/tree/stage3) [[8]](#footnote-91)

## Joining

Thus far, we have been dealing with a single node, which kind of misses the whole point of having a blockchain. So this section remedies this. We will add a new button the MainActivity to Join an existing blockchain. This will require discovering the network - we will just enter an IP address for the moment - although more complex schemes would be used in a production environment.

### Create Join Chat Activity

We create a new Activity: JoinChatActivity using the wizard at File > New > Activity > Empty Activity. Enter the name JoinChatActivity and the rest autocompletes.



### activity\_join\_chat.xml

First we set the layout in res/layout/activity\_join\_chat.xml. Overwrite all the contents of the file.

<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".JoinChatActivity">  
  
 <LinearLayout  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginStart="8dp"  
 android:layout\_marginLeft="8dp"  
 android:layout\_marginTop="8dp"  
 android:layout\_marginEnd="8dp"  
 android:layout\_marginRight="8dp"  
 android:layout\_marginBottom="8dp"  
 android:orientation="vertical"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent">  
  
 <EditText  
 android:id="@+id/editMoniker"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:ems="10"  
 android:hint="@string/moniker"  
 android:inputType="textPersonName" />  
  
 <EditText  
 android:id="@+id/editHost"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:ems="10"  
 android:hint="@string/host"  
 android:inputType="textUri" />  
  
 <Button  
 android:id="@+id/buttonJoin"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:onClick="joinChat"  
 android:text="@string/join" />  
 </LinearLayout>  
</androidx.constraintlayout.widget.ConstraintLayout>

This defines a screen where the user can enter a moniker, exactly as per the NewChatActivity. In contrast the NewChatActivity, there is an additional field to enter the address (IP/hostname) of an existing node on the network.

### strings.xml

We need to the add the following to res/values/strings.xml as they are used in the code changes below.

<string name="invalid\_hostname\_alert\_title"  
 >Invalid hostname</string>  
<string name="invalid\_hostname\_alert\_message"  
 >Please enter a valid hostname!</string>  
<string name="no\_hostname\_alert\_title">No hostname</string>  
<string name="no\_hostname\_alert\_message"  
 >Please enter a hostname!</string>  
<string name="peers\_error\_alert\_title"  
 >Unable to retrieve peers list</string>  
<string name="peers\_json\_error\_alert\_message"  
 >Did not receive a valid response from the host</string>  
<string name="peers\_connection\_error\_alert\_message">Failed to connect to host</string>  
<string name="peers\_timeout\_error\_alert\_message"  
 >Timed out waiting for host</string>  
<string name="peers\_unknown\_error\_alert\_message"  
 >Unknown error</string>  
<string name="loading\_title">Please wait...</string>  
<string name="loading\_message"  
 >Fetching peers list from host</string>  
<string name="join">Join</string>  
<string name="host">Hostname</string>

### JoinChatActivity.java

Now we need to add the Java source to JoinChatActivity.java overwriting the code that is already there.

package io.mosaicnetworks.myfirstapp;  
  
import androidx.appcompat.app.AlertDialog;  
import androidx.appcompat.app.AppCompatActivity;  
  
import android.app.ProgressDialog;  
import android.content.DialogInterface;  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.EditText;  
  
import androidx.annotation.StringRes;  
  
import java.util.List;  
  
import io.mosaicnetworks.babble.discovery.HttpPeerDiscoveryRequest;  
import io.mosaicnetworks.babble.discovery.Peer;  
import io.mosaicnetworks.babble.discovery.ResponseListener;  
  
public class JoinChatActivity extends AppCompatActivity  
 implements ResponseListener {  
  
 private ProgressDialog mLoadingDialog;  
 private String mMoniker;  
 private HttpPeerDiscoveryRequest mHttpGenesisPeerDiscoveryRequest;  
 private HttpPeerDiscoveryRequest mHttpCurrentPeerDiscoveryRequest;  
 private List<Peer> mGenesisPeers;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_join\_chat);  
 initLoadingDialog();  
 }  
  
 // called when the user presses the join chat button  
 public void joinChat(View view) {  
 //get moniker  
 EditText editText = findViewById(R.id.editMoniker);  
 mMoniker = editText.getText().toString();  
 if (mMoniker.isEmpty()) {  
 displayOkAlertDialog(R.string.no\_moniker\_alert\_title,  
 R.string.no\_moniker\_alert\_message);  
 return;  
 }  
  
 //get peer IP address  
 EditText editIP = findViewById(R.id.editHost);  
 final String peerIP = editIP.getText().toString();  
 if (peerIP.isEmpty()) {  
 displayOkAlertDialog(R.string.no\_hostname\_alert\_title,  
 R.string.no\_hostname\_alert\_message);  
 return;  
 }  
  
 getPeers(peerIP);  
 }  
  
 private void getPeers(final String peerIP) {  
 try {  
 mHttpGenesisPeerDiscoveryRequest =  
 HttpPeerDiscoveryRequest.createGenesisPeersRequest(  
 peerIP,  
 MessagingService.DEFAULT\_DISCOVERY\_PORT,  
 new ResponseListener() {  
 @Override  
 public void onReceivePeers(List<Peer> genesisPeers) {  
 mGenesisPeers = genesisPeers;  
  
 mHttpCurrentPeerDiscoveryRequest =  
 HttpPeerDiscoveryRequest.createCurrentPeersRequest(  
 peerIP,  
 MessagingService.DEFAULT\_DISCOVERY\_PORT,  
 JoinChatActivity.this,  
 JoinChatActivity.this);  
  
 mHttpCurrentPeerDiscoveryRequest.send();  
 }  
  
 @Override  
 public void onFailure(Error error) {  
 JoinChatActivity.this.onFailure(error);  
 }  
 }, this);  
 } catch (IllegalArgumentException ex) {  
 displayOkAlertDialog(  
 R.string.invalid\_hostname\_alert\_title,  
 R.string.invalid\_hostname\_alert\_message);  
 return;  
 }  
  
 mLoadingDialog.show();  
 mHttpGenesisPeerDiscoveryRequest.send();  
 }  
  
 @Override  
 public void onReceivePeers(List<Peer> currentPeers) {  
 MessagingService messagingService = MessagingService.getInstance();  
  
 try {  
 messagingService.configureJoin(mGenesisPeers, currentPeers,  
 mMoniker, Utils.getIPAddr(this));  
 } catch (IllegalStateException ex) {  
 //we tried to reconfigure before a leave completed  
 mLoadingDialog.dismiss();  
 displayOkAlertDialog(R.string.babble\_busy\_title,  
 R.string.babble\_busy\_message);  
 return;  
 }  
  
 mLoadingDialog.dismiss();  
 messagingService.start();  
 Intent intent = new Intent(JoinChatActivity.this,  
 ChatActivity.class);  
 intent.putExtra("MONIKER", mMoniker);  
 startActivity(intent);  
 }  
  
 @Override  
 public void onFailure(  
 io.mosaicnetworks.babble.discovery.ResponseListener.Error error) {  
   
 mLoadingDialog.dismiss();  
 int messageId;  
 switch (error) {  
 case INVALID\_JSON:  
 messageId = R.string.peers\_json\_error\_alert\_message;  
 break;  
 case CONNECTION\_ERROR:  
 messageId = R.string.peers\_connection\_error\_alert\_message;  
 break;  
 case TIMEOUT:  
 messageId = R.string.peers\_timeout\_error\_alert\_message;  
 break;  
 default:  
 messageId = R.string.peers\_unknown\_error\_alert\_message;  
 }  
 displayOkAlertDialog(R.string.peers\_error\_alert\_title, messageId);  
 }  
  
 private void initLoadingDialog() {  
 mLoadingDialog = new ProgressDialog(this);  
 mLoadingDialog.setProgressStyle(ProgressDialog.STYLE\_SPINNER);  
 mLoadingDialog.setTitle(R.string.loading\_title);  
 mLoadingDialog.setMessage(getString(R.string.loading\_message));  
 mLoadingDialog.setIndeterminate(true);  
 mLoadingDialog.setCanceledOnTouchOutside(false);  
 mLoadingDialog.setCancelable(true);  
  
 mLoadingDialog.setOnCancelListener(new DialogInterface.OnCancelListener(){  
 @Override  
 public void onCancel(DialogInterface dialog){  
 cancelRequets();  
 }});  
 }  
  
 private void displayOkAlertDialog(@StringRes int titleId,  
 @StringRes int messageId) {  
  
 AlertDialog alertDialog = new AlertDialog.Builder(this)  
 .setTitle(titleId)  
 .setMessage(messageId)  
 .setNeutralButton(R.string.ok\_button, null)  
 .create();  
 alertDialog.show();  
 }  
  
 private void cancelRequets() {  
 if (mHttpCurrentPeerDiscoveryRequest!=null) {  
 mHttpCurrentPeerDiscoveryRequest.cancel();  
 }  
  
 if (mHttpGenesisPeerDiscoveryRequest!=null) {  
 mHttpGenesisPeerDiscoveryRequest.cancel();  
 }  
 }  
  
 @Override  
 protected void onDestroy() {  
 cancelRequets();  
 super.onDestroy();  
 }  
  
}

There is a lot going in that code, so we will break it down into easier pieces. The JoinChatActivity is more complicated because it has to negotiate with another node and play nicely with it rather than just having the freedom to do its own thing that a NewChatActivity has. To join a network, our joining node needs to know the address of a node on the existing network. It then also needs the current peer set - so it know who to ask about the hashgraph history, and the genesis peer set (simply the peerset at the time that the network was instantiated). In a more complex example than this one, we would also need an initial state. By defining the initial state as blank, this app has circumvented that need, but in a more complex system utilising babble as a consensus engine, it would be required. Thus, monetd[[9]](#footnote-98) also requires a genesis.json file with a POA smart contract and initial Tenom assignments.

@Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_join\_chat);  
 initLoadingDialog();  
 }

The onCreate event sets up the UI and initialises a Dialog for later use.

public void joinChat(View view) {

JoinChat is involved on pressing the Join button. This sanity checks the user input and then calls:

getPeers(peerIP);

getPeers requests both peers lists from the node specified by the user. If successful, the function below is invoked:

public void onReceivePeers(List<Peer> currentPeers) {

Which contains the following lines. It uses the information retrieved from the existing node to configure the messagingService. The invocation is then exactly as per the NewChatActivity where the messagingService starts a babble node, and then invokes a ChatActivity.

messagingService.configureJoin(mGenesisPeers,   
 currentPeers, mMoniker, Utils.getIPAddr(this));  
  
...  
  
 messagingService.start();  
 Intent intent = new Intent(JoinChatActivity.this, ChatActivity.class);  
 intent.putExtra("MONIKER", mMoniker);  
 startActivity(intent);

### activity\_main.xml

We amend res/layout/activity\_main.xml to add a Join Button (the newChat button tag was already in the file):

<Button  
 android:id="@+id/button3"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:onClick="newChat"  
 android:text="@string/new\_chat" />  
  
 <Button  
 android:id="@+id/button4"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:onClick="joinChat"  
 android:text="@string/join" />

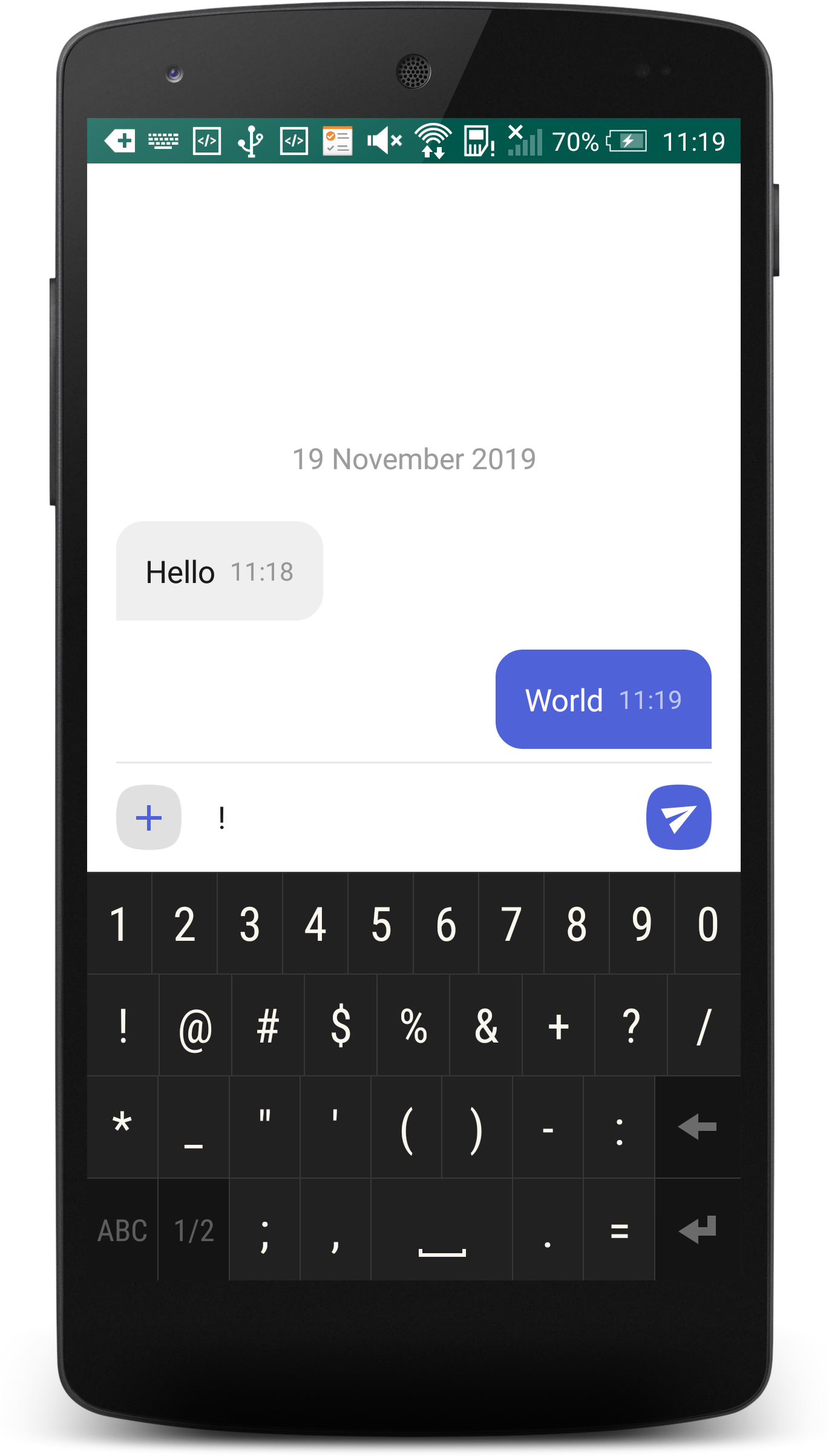
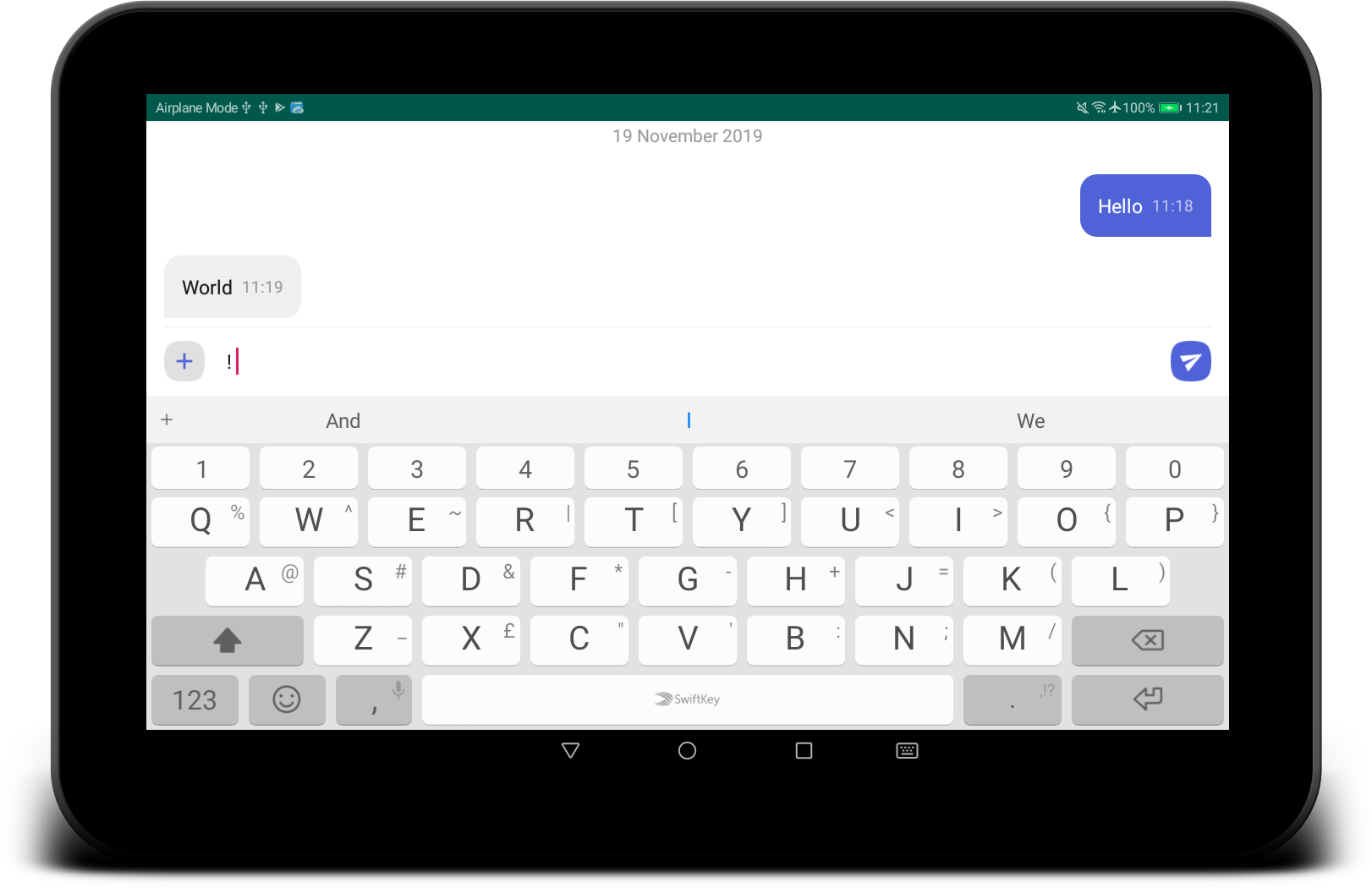
### MainActivity.java

Having added the Join button, we add the click handler to invoke the JoinChatActivity on clicking.

// called when the user presses the join chat button  
 public void joinChat(View view) {  
 Intent intent = new Intent(this, JoinChatActivity.class);  
 startActivity(intent);  
 }

### Build, Run and Test

Build your app and run it on 2 devices. You should now be able to start a chat on one and join with the other:

This project at this stage is available from github from [here](https://github.com/mosaicnetworks/babble-android-tutorial/tree/stage4) [[10]](#footnote-110)

1. The sample app is part of the babble-android library and is available from the [GitHub repo](https://github.com/mosaicnetworks/babble-android) [↑](#footnote-ref-22)
2. You can read more about AndroidX here: <https://android-developers.googleblog.com/2018/05/hello-world-androidx.html> [↑](#footnote-ref-36)
3. API version 19 is Android 4.4 (KitKat). In May 3.8% of devices were using version 18 or lower. Android 4.4 was released in 2013. Whilst it would be possible to code support for earlier versions, the existing code uses Android features introduced in Android 4.4. [↑](#footnote-ref-38)
4. This code is the stage1 branch at https://github.com/mosaicnetworks/babble-android-tutorial/tree/stage1 [↑](#footnote-ref-55)
5. [Android Studio Image Asset Studio](https://developer.android.com/studio/write/image-asset-studio) is described here: https://developer.android.com/studio/write/image-asset-studio [↑](#footnote-ref-57)
6. The sample app is part of the babble-android library and is available from the [GitHub repo](https://github.com/mosaicnetworks/babble-android) [↑](#footnote-ref-59)
7. This code is the stage2 branch at https://github.com/mosaicnetworks/babble-android-tutorial/tree/stage2 [↑](#footnote-ref-80)
8. This code is the stage3 branch at https://github.com/mosaicnetworks/babble-android-tutorial/tree/stage3 [↑](#footnote-ref-91)
9. [Monetd](https://github.com/mosaicnetworks/monetd) is the daemon component of the **Monet Toolchain**; a distributed smart-contract platform based on [EVM-Lite](https://github.com/mosaicnetworks/evm-lite) and [Babble](https://github.com/mosaicnetworks/babble).

   The **Monet Toolchain** underpins the [MONET Hub](https://monet.network/faq.html), but it is also available for use in other projects. You can read more about MONET in the [whitepaper](http://bit.ly/monet-whitepaper). [↑](#footnote-ref-98)
10. This code is the stage4 branch at https://github.com/mosaicnetworks/babble-android-tutorial/tree/stage4 [↑](#footnote-ref-110)