**Gradle**

|  |  |
| --- | --- |
|  | // Top-level build file where you can add configuration options common to all sub-projects/modules. |
|  |  |
|  | buildscript { |
|  | ext.kotlin\_version = '1.0.0' |
|  | repositories { |
|  | jcenter() |
|  | } |
|  | dependencies { |
|  | classpath 'com.android.tools.build:gradle:2.3.3' |
|  | classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin\_version" |
|  |  |
|  |  |
|  | classpath 'com.google.gms:google-services:3.0.0' |
|  | } |
|  | } |
|  |  |
|  | allprojects { |
|  | repositories { |
|  | jcenter() |
|  | } |
|  | } |
|  |  |
|  | task clean(type: Delete) { |
|  | delete rootProject.buildDir |
|  | } |

**AppLevel**

|  |  |
| --- | --- |
|  | apply plugin: 'com.android.application' |
|  | apply plugin: 'kotlin-android' |
|  | apply plugin: 'kotlin-android-extensions' |
|  |  |
|  | android { |
|  | compileSdkVersion 26 |
|  | buildToolsVersion "26.0.2" |
|  | defaultConfig { |
|  | applicationId "br.com.packapps.geofireapplication" |
|  | minSdkVersion 16 |
|  | targetSdkVersion 26 |
|  | versionCode 1 |
|  | versionName "1.0" |
|  | testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner" |
|  | } |
|  | buildTypes { |
|  | release { |
|  | minifyEnabled false |
|  | proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro' |
|  | } |
|  | } |
|  | } |
|  |  |
|  | dependencies { |
|  | compile fileTree(dir: 'libs', include: ['\*.jar']) |
|  |  |
|  | compile "org.jetbrains.kotlin:kotlin-stdlib:$kotlin\_version" |
|  | compile 'com.android.support:appcompat-v7:26.+' |
|  | compile 'com.android.support.constraint:constraint-layout:1.0.2' |
|  |  |
|  | compile 'com.google.firebase:firebase-database:10.0.1' |
|  | } |
|  | repositories { |
|  | mavenCentral() |
|  | } |
|  |  |
|  | apply plugin: 'com.google.gms.google-services' |

**Salvando Localização**

|  |
| --- |
| class MainActivity : AppCompatActivity() { |
|  |  |
|  | var timeMillisId : Long = 0 |
|  |  |
|  | override fun onCreate(savedInstanceState: Bundle?) { |
|  | super.onCreate(savedInstanceState) |
|  | setContentView(R.layout.activity\_main) |
|  |  |
|  | val db : DatabaseReference = FirebaseDatabase.getInstance().getReference("geo/providers") |
|  | val geofire = GeoFire(db) |
|  |  |
|  | //### set Location |
|  | timeMillisId = System.currentTimeMillis() |
|  | geofire.setLocation(timeMillisId.toString(), GeoLocation(-1.3904518, -48.4673762), GeoFire.CompletionListener { |
|  | key, error -> |
|  | if (error == null) |
|  | Log.i("TAG", "geo added successful: " + key) |
|  | else |
|  | Log.i("TAG", "geo added error: " + error.message) |
|  | }) |
|  |  |
|  | //### listener for disconnect |
|  | db.child(timeMillisId.toString()).onDisconnect().removeValue() |
|  |  |
|  |  |
|  |  |
|  | } |

No nosso exemplo vamos simular o uso de um APP tipo Uber, YetGo, etc, que precisa disponibilizar a seus clientes a localização dos carros(providers).

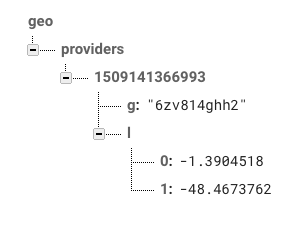
* Na*linha 9*, nada de mais, obtemos a instância do Firebase Database, apontando para o end-point **"geo/providers"**
* Na *linha 10*, começamos com o **GeoFire**. Note que passamos como argumento nossa referência **db**.
* Na *linha 14* começamos nosso trabalho de salvar nossos dados. Usando o método *setLocation(…)* com três parâmetros:

geofire.setLocation(  
 "**Nome-do-seu-identificador**",   
 um objeto **GeoLocation** que contém as coordenadas,   
 e um **listener** para ouvir o retorno para   
 verificar erro ou sucesso)

* *Linha 13*, criamos uma chave usando *System.currentTimeMillis()* .

*O Firebase possui um método muito útil para se trabalhar com listas de dados. Neste nosso caso, poderíamos simplesmente chamar****db.push()****que um novo item seria adicionado a nossa****DatabaseReference****. Para adicionar este item e ao mesmo tempo saber o valor de sua key, é só chamar a propriedade***key***, ficando assim:****db.push().key****. —e esse passaria a ser nosso identificador na linha 13.*

E nosso modelo de dados no Firebase ficou assim:



*Com exceção da chave 1509141366993, que foi criado por nós mesmos, os outros pares de chaves são padrão do objeto GeoFire.*

* E na*linha 23*, removemos nossa referência com;

db!!.child("Nome-do-seu-identificador").onDisconnect().removeValue()

**Ouvindo os dados**

Para fazermos a pesquisa dos dados e ficarmos *ouvindo* isso em *real-time*, o pacote *geofire* possui um objeto chamado **GeoQuery** que obtemos como retorno chamando o método de **GeoFire**:  
  
queryAtLocation(location: GeoLocation, radiusKm : Long)

|  |
| --- |
| class MainActivity : AppCompatActivity() { |
|  | var db : DatabaseReference? = null |
|  | var geofire : GeoFire? = null |
|  |  |
|  | var timeMillisId : Long = 0 |
|  |  |
|  | override fun onCreate(savedInstanceState: Bundle?) { |
|  |  |
|  | ... |
|  |  |
|  | //### Getting providers |
|  | val geoQuery : GeoQuery = geofire!!.queryAtLocation(GeoLocation(-1.4580218, -48.4968418), 7.0) |
|  |  |
|  | geoQuery.addGeoQueryEventListener(object : GeoQueryEventListener { |
|  | override fun onKeyEntered(key: String, location: GeoLocation) { |
|  | Log.i("TAG", String.format("Provider %s is within your search range [%f,%f]", key, location.latitude, location.longitude)) |
|  | } |
|  |  |
|  | override fun onKeyExited(key: String) { |
|  | Log.i("TAG", String.format("Provider %s is no longer in the search area", key)) |
|  | } |
|  |  |
|  | override fun onKeyMoved(key: String, location: GeoLocation) { |
|  | Log.i("TAG", String.format("Provider %s moved within the search area to [%f,%f]", key, location.latitude, location.longitude)) |
|  | } |
|  |  |
|  | override fun onGeoQueryReady() { |
|  | Log.i("TAG", "onGeoQueryReady") |
|  | } |
|  |  |
|  | override fun onGeoQueryError(error: DatabaseError) { |
|  | Log.e("TAG", "error: " + error) |
|  | } |
|  | }) |
|  |
|  |
|  | ... |
|  |  |
|  | } |

Como explicado anteriormente sobre o método queryAtLocation(…), estamos passando, na linha 12, uma coordenada simulando nossa posição atual no GPS, e o valor 7.0 como sendo o raio de alcance que queremos escutar no resultado do real-time.

Na *linha 14*, já em nossa instância de **GeoQuery**, chamamos o método

addGeoQueryEventListener(callback : GeoQueryEventListener)

O *callback* **GeoQueryEventListener** possui cinco métodos que devem ser implementados.

**Um pequeno problema**

A localização do provider é o suficiente para nosso cliente ? E se quisermos saber a cor, a placa do carro, etc ?

Infelizmente a nossa instância de **GeoFire** não nos permite aninhar dados com os dados padrões deste objeto.

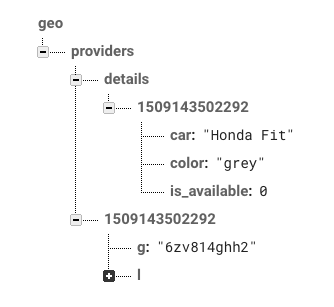
*Você pode até tentar fazer isso direto pela instância de****DatabaseReference****apontando para a nossa chave — Acha que não tentei? Não rola! — . Os dados serão totalmente sobrescritos e você perderá as informações de location.*

Uma saída pra isso é você criar uma abertura de árvore nova — eu chamei de *details a minha* — para salvar dados extras vinculados a nossa chave de location.

O lugar ideal para fazer isso é no *callback* do momento que estamos salvando a localização do provider no database. Veja:

|  |
| --- |
| ... |
|  | private fun setLocationInDatabase(key : String) { |
|  | geofire!!.setLocation(key, GeoLocation(-1.3904518, -48.4673762), GeoFire.CompletionListener { key, error -> |
|  | if (error == null) { |
|  | Log.i("TAG", "geo added successful: " + key) |
|  | //Save detail |
|  | val moreDataProvider: HashMap<String, Any> = hashMapOf("car" to "Honda Fit", "color" to "grey", "is\_available" to 0) |
|  | db!!.child("details").child(key).setValue(moreDataProvider) |
|  |  |
|  | }else { |
|  | Log.i("TAG", "geo added error: " + error.message) |
|  | } |
|  | }) |
|  | } |
|  | ... |

A mágica está na *linha 7 e 8*, foi a única alteração que precisamos fazer, pois em caso de sucesso, criamos uma nova árvore com *child("details")* onde salvamos mais detalhes dos providers. Dentro de "details" temos as mesmas chaves location dos providers (*child(key)*). E por fim passamos nossos dados extras presentes num HashMap (*setValue(moreDataProvider)*)



SELECT \*, (6371 \*

acos(

cos(radians(-19.83996)) \*

cos(radians(lat)) \*

cos(radians(-43.94910) - radians(lng)) +

sin(radians(-19.83996)) \*

sin(radians(lat))

)) AS distance

FROM tabela HAVING distance <= 5

Variando...

SELECT \*,(RAIO\_TERRESTRE \*

acos(

cos(radians(PARAMETRO\_LATITUDE)) \*

cos(radians(COLUNA\_LATITUDE)) \*

cos(radians(PARAMETRO\_LONGITUDE) - radians(COLUNA\_LONGITUDE)) +

sin(radians(PARAMETRO\_LATITUDE)) \*

sin(radians(COLUNA\_LATITUDE))

)) AS CAMPOLATITUDE

FROM TABELA HAVING CAMPOLATITUDE <= KM

Link:

<https://medium.com/@paulo_linhares/firebase-geofire-obtendo-coordenadas-dentro-de-um-determinado-raio-geogr%C3%A1fico-app-android-a57c0202a5ed>

<https://pt.stackoverflow.com/questions/55669/identificar-se-conjunto-de-coordenadas-est%C3%A1-dentro-de-um-raio-em-android/55710>