


Firestore Database


Realtime
Database









Storing & Retrieving
placemarks to/from Firestore

Realtime Database

 **Firebase**

[Project Overview](#) 

Develop

-  Authentication
-  **Database**
-  Storage
-  Hosting
-  Functions
-  ML Kit



Quality
Crashlytics, Performance, Test Lab

Analytics
Dashboard, Events, Conversions, Au...


Grow
Predictions, A/B Testing, Cloud Mes...


Spark


placemark-lab ▾ Database


[Go to docs](#)  


Learn more

 Find out if Cloud Firestore is right for you
Compare databases

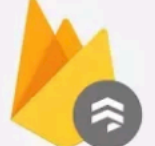
 How do I get started?
View the docs

 How much will Cloud Firestore cost?
View pricing


 What can Cloud Firestore do for me?
Learn more



Introducing Cloud Firestore [Watch later](#) [Share](#)

 **Cloud Firestore**

Or choose Realtime Database



Realtime Database

Firebase's original database. Like Cloud Firestore, it supports realtime data synchronization.

[View the docs](#) [Learn more](#)

Create database

Security rules for Realtime Database

Once you have defined your data structure you will have to write rules to secure your data.
[Learn more](#)

☐

Start in **locked mode**

Make your database private by denying all reads and writes

☒

Start in **test mode**

Get set up quickly by allowing all reads and writes to your database

```
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

!

Anyone with your database reference will be able to read or write to your database

Cancel

Enable

https://console.firebase.google.com

placemark-lab

Database

Realtime Database

Data

Rules

Backups

Usage

https://placemark-lab.firebaseio.com/



! Your security rules are defined as public, so anyone can steal, modify, or delete data in your database

Learn more

Dismiss

placemark-lab: null

Android Studio

Assistant  

← **Firebase** > Realtime Database

Save and retrieve data

Our cloud database stays synced to all connected clients in realtime and remains available when your app goes offline. Data is stored in a JSON tree structure rather than a table, eliminating the need for complex SQL queries.

[Launch in browser](#)

1

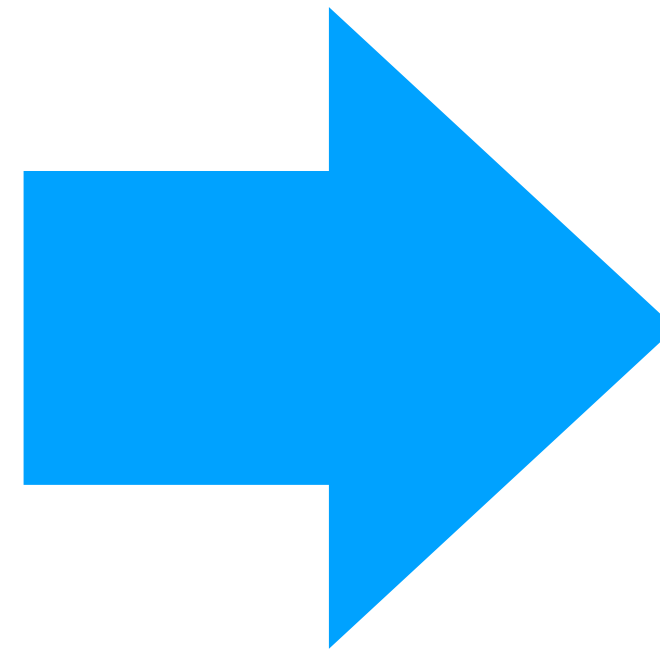
Connect your app to Firebase

✓ Connected

2

Add the Realtime Database to your app

Add the Realtime Database to your app



Updates
app/google-services.json

```
{
  "project_info": {
    "project_number": "4283XXXX",
    "firebase_url": "https://placemark-XXXX.firebaseio.com",
    "project_id": "placemark-XXXd",
  },
  "client": [
    {
      "client_info": {
        "mobilesdk_app_id": "1:428338485028:android:634c4XXXce143",
        "android_client_info": {
          "package_name": "org.wit.placemark"
        }
      },
      "oauth_client": [
        {
          "client_id": "4283XXXX028-ntqXXXXXXXXXl9ot6ok3r.apps.googleusercontent.com",
          "client_type": 1,
          "android_info": {
            "package_name": "org.wit.placemark",
            "certificate_hash": "bcaa865ad78XXXXXXXXX731db4da8b"
          }
        },
        {
          "client_id": "42833848XXXXX5cup7XXXXXXk8s.apps.googleusercontent.com",
          "client_type": 3
        }
      ],
      "api_key": [
        {
          "current_key": "AIzaSyBXXXXXXXXXXXXXoTeWhTqfKxbI"
        }
      ],
      "services": {
        "analytics_service": {
          "status": 1
        },
        "appinvite_service": {
          "status": 2,
          "other_platform_oauth_client": [
            {
              "client_id": "428338XXXXXXXXXXXXXXXXX1e4kk8s.apps.googleusercontent.com",
              "client_type": 3
            }
          ]
        },
        "ads_service": {
          "status": 2
        }
      }
    }
  ],
  "configuration_version": "1"
}
```

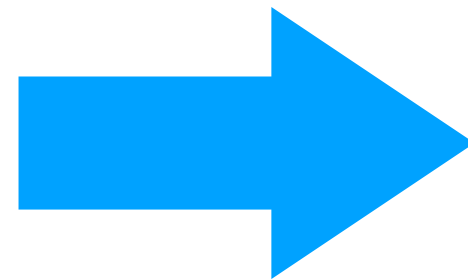
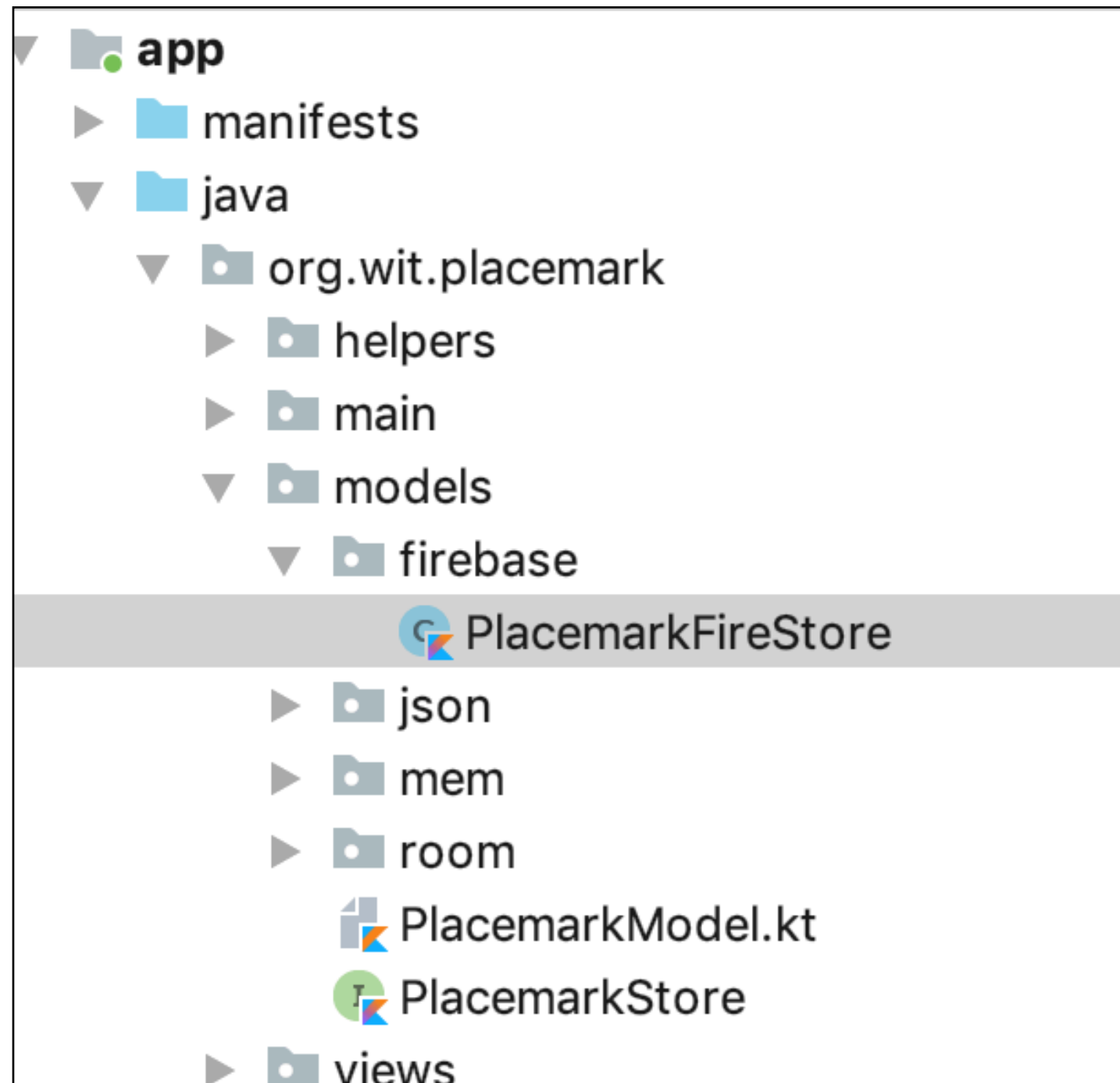

PlacemarkModel

```
@Parcelize
@Entity
data class PlacemarkModel(@PrimaryKey(autoGenerate = true) var id: Long = 0,
    var fbId : String = "",
    var title: String = "",
    var description: String = "",
    var image: String = "",
    @Embedded var location : Location = Location()): Parcelable

@Parcelize
data class Location(var lat: Double = 0.0,
    var lng: Double = 0.0,
    var zoom: Float = 0f) : Parcelable
```

New Field: fbId - used to store Firebase key (a string)
Otherwise, model unchanged

PlacemarkFirestore



```
class PlacemarkFirestore(val context: Context) : PlacemarkStore, AnkoLogger {

    val placemarks = ArrayList<PlacemarkModel>()
    lateinit var userId: String
    lateinit var db: DatabaseReference

    suspend override fun findAll(): List<PlacemarkModel> {
        return placemarks
    }

    suspend override fun findById(id: Long): PlacemarkModel? {
        val foundPlacemark: PlacemarkModel? = placemarks.find { p -> p.id == id }
        return foundPlacemark
    }

    suspend override fun create(placemark: PlacemarkModel) {
        val key = db.child("users").child(userId).child("placemarks").push().key
        placemark.fbId = key!!
        placemarks.add(placemark)
        db.child("users").child(userId).child("placemarks").child(key).setValue(placemark)
    }

    suspend override fun update(placemark: PlacemarkModel) {
        var foundPlacemark: PlacemarkModel? = placemarks.find { p -> p.fbId == placemark.fbId }
        if (foundPlacemark != null) {
            foundPlacemark.title = placemark.title
            foundPlacemark.description = placemark.description
            foundPlacemark.image = placemark.image
            foundPlacemark.location = placemark.location
        }

        db.child("users").child(userId).child("placemarks").child(placemark.fbId).setValue(placemark)
    }

    suspend override fun delete(placemark: PlacemarkModel) {
        db.child("users").child(userId).child("placemarks").child(placemark.fbId).removeValue()
        placemarks.remove(placemark)
    }

    override fun clear() {
        placemarks.clear()
    }

    fun fetchPlacemarks(placemarksReady: () -> Unit) {
        val valueEventListener = object : ValueEventListener {
            override fun onCancelled(error: DatabaseError) {
            }
            override fun onDataChange(dataSnapshot: DataSnapshot) {
                dataSnapshot.children.mapNotNullTo(placemarks) { it.getValue<PlacemarkModel>(PlacemarkModel::class.java) }
                placemarksReady()
            }
        }
        userId = FirebaseAuth.getInstance().currentUser!!.uid
        db = FirebaseDatabase.getInstance().reference
        placemarks.clear()
        db.child("users").child(userId).child("placemarks").addListenerForSingleValueEvent(valueEventListener)
    }
}
```

PlacemarkFirestore - Initialisation

```
class PlacemarkFirestore(val context: Context) : PlacemarkStore, AnkoLogger {  
  
    val placemarks = ArrayList<PlacemarkModel>()  
    lateinit var userId: String  
    lateinit var db: DatabaseReference  
  
    suspend override fun findAll(): List<PlacemarkModel> {  
        return placemarks  
    }  
  
    suspend override fun findById(id: Long): PlacemarkModel? {  
        val foundPlacemark: PlacemarkModel? = placemarks.find { p -> p.id == id }  
        return foundPlacemark  
    }  
  
    ...  
  
    fun fetchPlacemarks(...) {  
        userId = FirebaseAuth.getInstance().currentUser!!.uid  
        db = FirebaseDatabase.getInstance().reference  
        ...  
    }  
    ...  
}
```

Firestore UserID
(from Auth)

Firestore Database
Reference

Database Structure

Database

Realtime Database

DataRulesBackupsUsage

https://placemark-222108.firebaseio.com/

+

−

⋮

⚠

Your security rules are defined as public, so anyone can steal, modify, or delete data in your database

[Learn more](#)

[Dismiss](#)

placemark-222108

−

users

−

9VCoy8TSQCU08vMKAMg1qYfi2RT2

−

placemarks

+

-LRajxT1JeBmOldBJLx-

+

-LRak-ZGQC28xel8Imbh

+

-LRalieCFxhl-Nng3nu3

−

-LRbdKYneAL81oF5WKEk

description: "\"Ecellent Location\""

fbId: "-LRbdKYneAL81oF5WKEk"

id: 0

image: "https://firebasestorage.googleapis.com/v0/b/pla..."

−

location

lat: 52.245695

lng: -7.1391017

zoom: 15

title: "asd"

+

fUZhVTJ1ZhROOU0I1CenEzYWLwo2

Application Database

placemark-222108

users

9VCoy8TSQCU08vMKAMg1qYfi2RT2

placemarks

-LRajxT1JeBm0ldBJLx-

-LRak-ZGQC28xel8Imbh

-LRalieCFxhl-Nng3nu3

-LRbdKYneAL81oF5WKEk

description: "\"Ecellent Location"

fbId: "-LRbdKYneAL81oF5WKEk"

id: 0

image: "https://firebasestorage.googleapis.com/v0/b/pla..."

location

lat: 52.245695

lng: -7.1391017

zoom: 15

title: "asd"

fUZhVTJ1ZhR00U0I1CenEzYWLwo2

Application
Database

Collection of all
Users

placemark-222108



users



9VCoy8TSQCU08vMKAMg1qYfi2RT2



placemarks



-LRajxT1JeBm0ldBJLx-



-LRak-ZGQC28xel8Imbh



-LRalieCFxhl-Nng3nu3



-LRbdKYneAL81oF5WKEk

description: "\"Ecellent Location"

fbId: "-LRbdKYneAL81oF5WKEk"

id: 0

image: "https://firebasestorage.googleapis.com/v0/b/pla..."



location

lat: 52.245695

lng: -7.1391017

zoom: 15

title: "asd"



fUZhVTJ1ZhR00U0I1CenEzYWLwo2

Application
Database

Collection of all
Users

Individual user
(based in Auth ID)

placemark-222108

users

9VCoy8TSQCU08vMKAMg1qYfi2RT2

placemarks

-LRajxT1JeBm0ldBJLx-

-LRak-ZGQC28xel8Imbh

-LRalieCFxhl-Nng3nu3

-LRbdKYneAL81oF5WKEk

description: "\"Ecellent Location"

fbId: "-LRbdKYneAL81oF5WKEk"

id: 0

image: "https://firebasestorage.googleapis.com/v0/b/pla..."

location

lat: 52.245695

lng: -7.1391017

zoom: 15

title: "asd"

fUZhVTJ1ZhR00U0I1CenEzYWLwo2

Application
Database

Collection of all
Users

Individual user
(based in Auth ID)

This Users'
placemark
collection



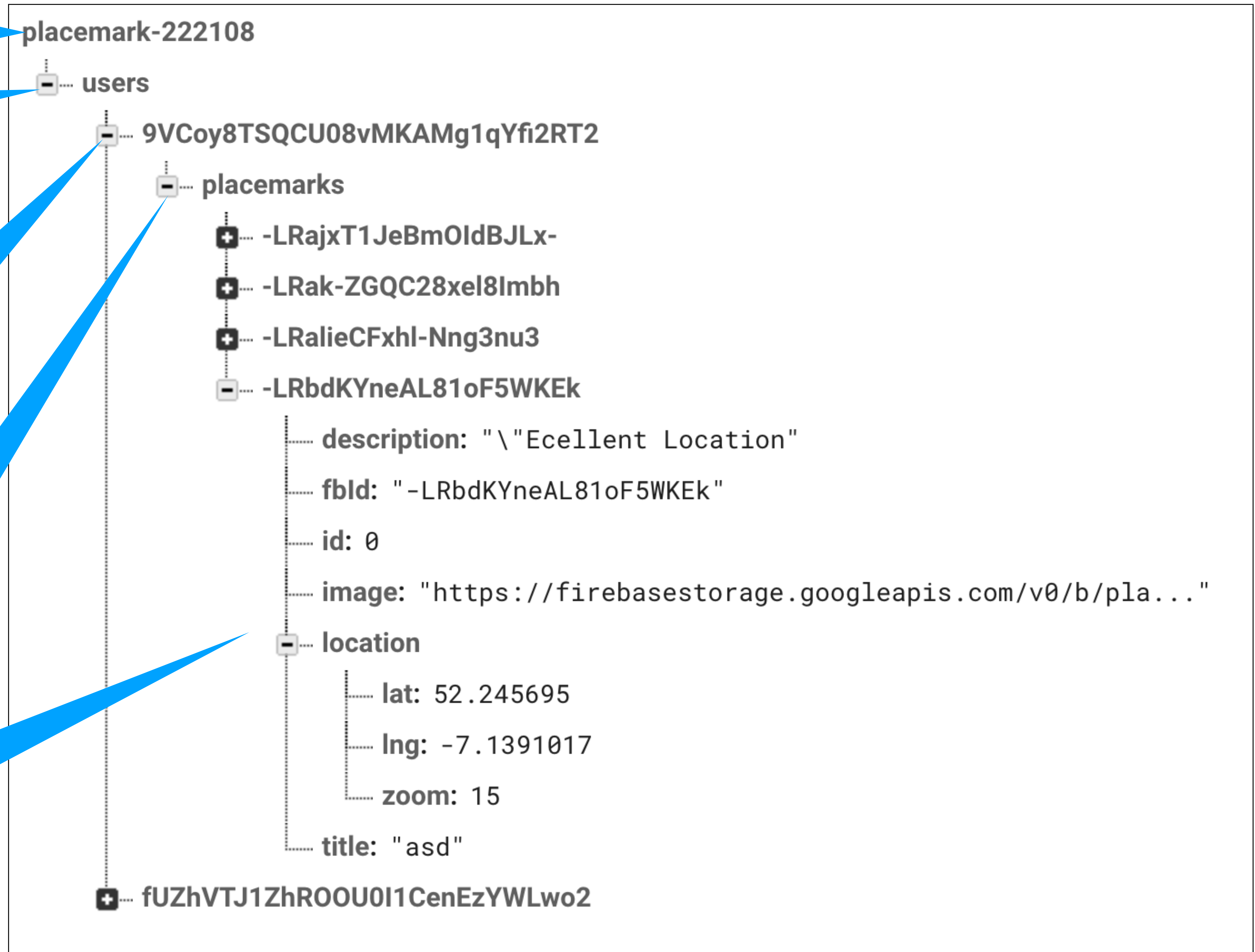
Application
Database

Collection of all
Users

Individual user
(based in Auth ID)

This Users'
placemark
collection

Individual
Placemark



PlacemarkFirestore - Create

Create a new Placemark object in the Database

Retain firebase key in place mark object

Populate the object with Placemark details

```
class PlacemarkFirestore(val context: Context) : PlacemarkStore, AnkoLogger {  
  
    val placemarks = ArrayList<PlacemarkModel>()  
    lateinit var userId: String  
    lateinit var db: DatabaseReference  
  
    suspend override fun create(placemark: PlacemarkModel) {  
  
        val key = db.child("users").child(userId).child("placemarks").push().key  
        key?.let {  
            placemark.fbId = key  
            placemarks.add(placemark)  
            db.child("users").child(userId).child("placemarks").child(key).setValue(placemark)  
        }  
    }  
    ...  
}
```

Keep local copy of Placemark in placemarks array

PlacemarkFirestore - update

Update
pacemark in
local array

```
suspend override fun update(placemark: PlacemarkModel) {  
    var foundPlacemark: PlacemarkModel? = placemarks.find { p -> p.fbId == placemark.fbId }  
    if (foundPlacemark != null) {  
        foundPlacemark.title = placemark.title  
        foundPlacemark.description = placemark.description  
        foundPlacemark.image = placemark.image  
        foundPlacemark.location = placemark.location  
    }  
  
    db.child("users").child(userId).child("placemarks").child(placemark.fbId).setValue(placemark)  
}
```

Replace placemark in database with
new values

PlacemarkFirestore - delete

```
suspend override fun delete(placemark: PlacemarkModel) {  
    db.child("users").child(userId).child("placemarks").child(placemark.fbId).removeValue()  
    placemarks.remove(placemark)  
}
```


PlacemarkFirestore - fetchPlacemarks

```
fun fetchPlacemarks(placemarksReady: () -> Unit) {  
    val valueEventListener = object : ValueEventListener {  
        override fun onCancelled(dataSnapshot: DatabaseError) {  
            // Error connecting to database  
        }  
        override fun onDataChange(dataSnapshot: DataSnapshot) {  
            dataSnapshot!!.children.mapNotNullTo(placemarks) {  
                it.getValue<PlacemarkModel>(PlacemarkModel::class.java)  
            }  
            placemarksReady()  
        }  
    }  
}  
  
userId = FirebaseAuth.getInstance().currentUser!!.uid  
db = FirebaseDatabase.getInstance().reference  
placemarks.clear()  
  
db.child("users").child(userId).child("placemarks").addListenerForSingleValueEvent(valueEventListener)  
}
```

PlacemarkFirestore - fetchPlacemarks

```
fun fetchPlacemarks(placemarksReady: () -> Unit) {  
    val valueEventListener = object : ValueEventListener {  
        override fun onCancelled(dataSnapshot: DatabaseError) {  
            // Error connecting to database  
        }  
        override fun onDataChange(dataSnapshot: DataSnapshot) {  
            dataSnapshot!!.children.mapNotNullTo(placemarks) {  
                it.getValue<PlacemarkModel>(PlacemarkModel::class.java)  
            }  
            placemarksReady()  
        }  
    }  
  
    userId = FirebaseAuth.getInstance().currentUser!!.uid  
    db = FirebaseDatabase.getInstance().reference  
    placemarks.clear()  
  
    db.child("users").child(userId).child("placemarks").addListenerForSingleValueEvent(valueEventListener)  
}
```

Listener Callback
object for Database
updates

Listen for single update - in this case will be
triggered with complete placemark collection

PlacemarkFirestore - fetchPlacemarks

```
fun fetchPlacemarks(placemarksReady: () -> Unit) {  
    val valueEventListener = object : ValueEventListener {  
        override fun onCancelled(dataSnapshot: DatabaseError) {  
            // Error connecting to database  
        }  
        override fun onDataChange(dataSnapshot: DataSnapshot) {  
            dataSnapshot!!.children.mapNotNullTo(placemarks) {  
                it.getValue<PlacemarkModel>(PlacemarkModel::class.java)  
            }  
            placemarksReady()  
        }  
    }  
}  
  
userId = FirebaseAuth.getInstance().currentUser!!.uid  
db = FirebaseDatabase.getInstance().reference  
placemarks.clear()  
  
db.child("users").child(userId).child("placemarks").addListenerForSingleValueEvent(valueEventListener)  
}
```

Copy retrieved
peacemakers to
local array

Lambda we will call
when placemarks have
been retrieved

PlacemarkFirestore - fetchPlacemarks

```
fun fetchPlacemarks(placemarksReady: () -> Unit) {
```

```
    val valueEventListener = object : ValueEventListener {  
        override fun onCancelled(dataSnapshot: DatabaseError) {  
            // Error connecting to database  
        }  
        override fun onDataChange(dataSnapshot: DataSnapshot) {  
            dataSnapshot!!.children.mapNotNullTo(placemarks) {  
                it.getValue<PlacemarkModel>(PlacemarkModel::class.java)  
            }  
            placemarksReady()  
        }  
    }
```

Trigger lambda - as place
marks have been retrieved

```
    userId = FirebaseAuth.getInstance().currentUser!!.uid  
    db = FirebaseDatabase.getInstance().reference  
    placemarks.clear()
```

```
    db.child("users").child(userId).child("placemarks").addListenerForSingleValueEvent(valueEventListener)  
}
```


Lambda we will call
when placemarks have
been retrieved

PlacemarkFirestore - fetchPlacemarks

```
fun fetchPlacemarks(placemarksReady: () -> Unit) {  
    val valueEventListener = object : ValueEventListener {  
        override fun onCancelled(databaseError: DatabaseError) {  
            // Error connecting to database  
        }  
        override fun onDataChange(dataSnapshot: DataSnapshot) {  
            dataSnapshot!!.children.mapNotNullTo(placemarks) {  
                it.getValue<PlacemarkModel>(PlacemarkModel::class.java)  
            }  
            placemarksReady()  
        }  
    }  
  
    userId = FirebaseAuth.getInstance().currentUser!!.uid  
    db = FirebaseDatabase.getInstance().reference  
    placemarks.clear()  
  
    db.child("users").child(userId).child("placemarks").addListenerForSingleValueEvent(valueEventListener)  
}
```

Copy retrieved
peacemakers to
local array

Trigger lambda - as place
marks have been retrieved

Listener Callback
object for Database
updates

Listen for single update - in this case will be
triggered with complete placemark collection

LoginPresenter

```
class LoginPresenter(view: BaseView) : BasePresenter(view) {

    var auth: FirebaseAuth = FirebaseAuth.getInstance()
    var firestore: PlacemarkFirestore? = null

    init {
        if (app.placemarks is PlacemarkFirestore) {
            firestore = app.placemarks as PlacemarkFirestore
        }
    }

    fun doLogin(email: String, password: String) {
        view?.showProgress()
        auth.signInWithEmailAndPassword(email, password).addOnCompleteListener(view!!) { task ->
            if (task.isSuccessful) {
                if (firestore != null) {
                    firestore!!.fetchPlacemarks {
                        view?.hideProgress()
                        view?.navigateTo(VIEW.LIST)
                    }
                } else {
                    view?.hideProgress()
                    view?.navigateTo(VIEW.LIST)
                }
            } else {
                view?.hideProgress()
                view?.toast("Sign Up Failed: ${task.exception?.message}")
            }
        }
    }

    ...
}
```

LoginPresenter : doLogin

```
fun doLogin(email: String, password: String) {  
    view?.showProgress()  
    auth.signInWithEmailAndPassword(email, password).addOnCompleteListener(view!!) { task ->  
        if (task.isSuccessful) {  
            if (fireStore != null) {  
                firestore!!.fetchPlacemarks {  
                    view?.hideProgress()  
                    view?.navigateTo(VIEW.LIST)  
                }  
            } else {  
                view?.hideProgress()  
                view?.navigateTo(VIEW.LIST)  
            }  
        } else {  
            view?.hideProgress()  
            view?.toast("Sign Up Failed: ${task.exception?.message}")  
        }  
    }  
}
```

lambda to be called when
place marks have been
retrieved

LoginPresenter -> PlacemarkStore -> LoginPresenter

```
fun fetchPlacemarks(placemarksReady: () -> Unit) {  
    val valueEventListener = object : ValueEventListener {  
        override fun onCancelled(dataSnapshot: DataSnapshot) {  
            // Error connecting to database  
        }  
        override fun onDataChange(dataSnapshot: DataSnapshot) {  
            dataSnapshot!!.children.mapNotNullTo(placemarksReady) {  
                it.getValue<PlacemarkModel>(PlacemarkModel::class.java)  
            }  
            placemarksReady()  
        }  
    }  
}
```

```
userId = FirebaseAuth.getInstance().currentUser?.uid  
db = FirebaseDatabase.getInstance().reference  
placemarks.clear()
```

```
db.child("users").child(userId).child("placemarks")
```

```
fun doLogin(email: String, password: String) {  
    view?.showProgress()  
    auth.signInWithEmailAndPassword(email, password).addOnCompleteListener(task) {  
        if (task.isSuccessful) {  
            if (fireStore != null) {  
                fireStore!!.fetchPlacemarks {  
                    view?.hideProgress()  
                    view?.navigateTo(VIEW.LIST)  
                }  
            } else {  
                view?.hideProgress()  
                view?.navigateTo(VIEW.LIST)  
            }  
        } else {  
            view?.hideProgress()  
            view?.toast("Sign Up Failed: ${task.exception?.message}")  
        }  
    }  
}
```


Supplementary Approach: Enabling Offline Capabilities

Firebase apps automatically handle temporary network interruptions.

Cached data is available while offline and Firebase resends any writes when network connectivity is restored.

```
FirebaseDatabase.getInstance().setPersistenceEnabled(true);
```

Persistence Behaviour

Keeping Data Fresh

Querying Data Offline

Handling Transactions Offline

Managing Presence

Detecting Connection State

<https://firebase.google.com/docs/database/android/offline-capabilities>

Enabling Offline Capabilities on Android



Contents

Disk Persistence

Persistence Behavior

Keeping Data Fresh

Querying Data Offline

...

Firebase applications work even if your app temporarily loses its network connection. In addition, Firebase provides tools for persisting data locally, managing presence, and handling latency.

Disk Persistence



Firebase apps automatically handle temporary network interruptions. Cached data is available while offline and Firebase resends any writes when network connectivity is restored.

When you enable disk persistence, your app writes the data locally to the device so your app can maintain state while offline, even if the user or operating system restarts the app.

You can enable disk persistence with just one line of code.

```
FirebaseDatabase.getInstance().setPersistenceEnabled(true);
```



Persistence Behavior

By enabling persistence, any data that the Firebase Realtime Database client would sync while online persists to disk and is available offline, even when the user or operating system restarts the app. This means your app works as it would online by using the local data stored in the cache. Listener callbacks will continue to fire for local updates.

The Firebase Realtime Database client automatically keeps a queue of all write operations that are performed while your app is offline. When persistence is enabled, this queue is also persisted to disk so all of your writes are available when the user or operating system restarts the app. When the app regains connectivity, all of the operations are sent to the Firebase Realtime Database server.

If your app uses [Firebase Authentication](#), the Firebase Realtime Database client persists the user's authentication token across app restarts. If the auth token expires while your app is offline, the client pauses write operations until your app re-authenticates the user, otherwise the write operations might fail due to security rules.

Keeping Data Fresh

The Firebase Realtime Database synchronizes and stores a local copy of the data for active listeners. In addition, you can keep specific locations in sync.

```
DatabaseReference scoresRef = FirebaseDatabase.getInstance().getReference("scores");  
scoresRef.keepSynced(true);
```

