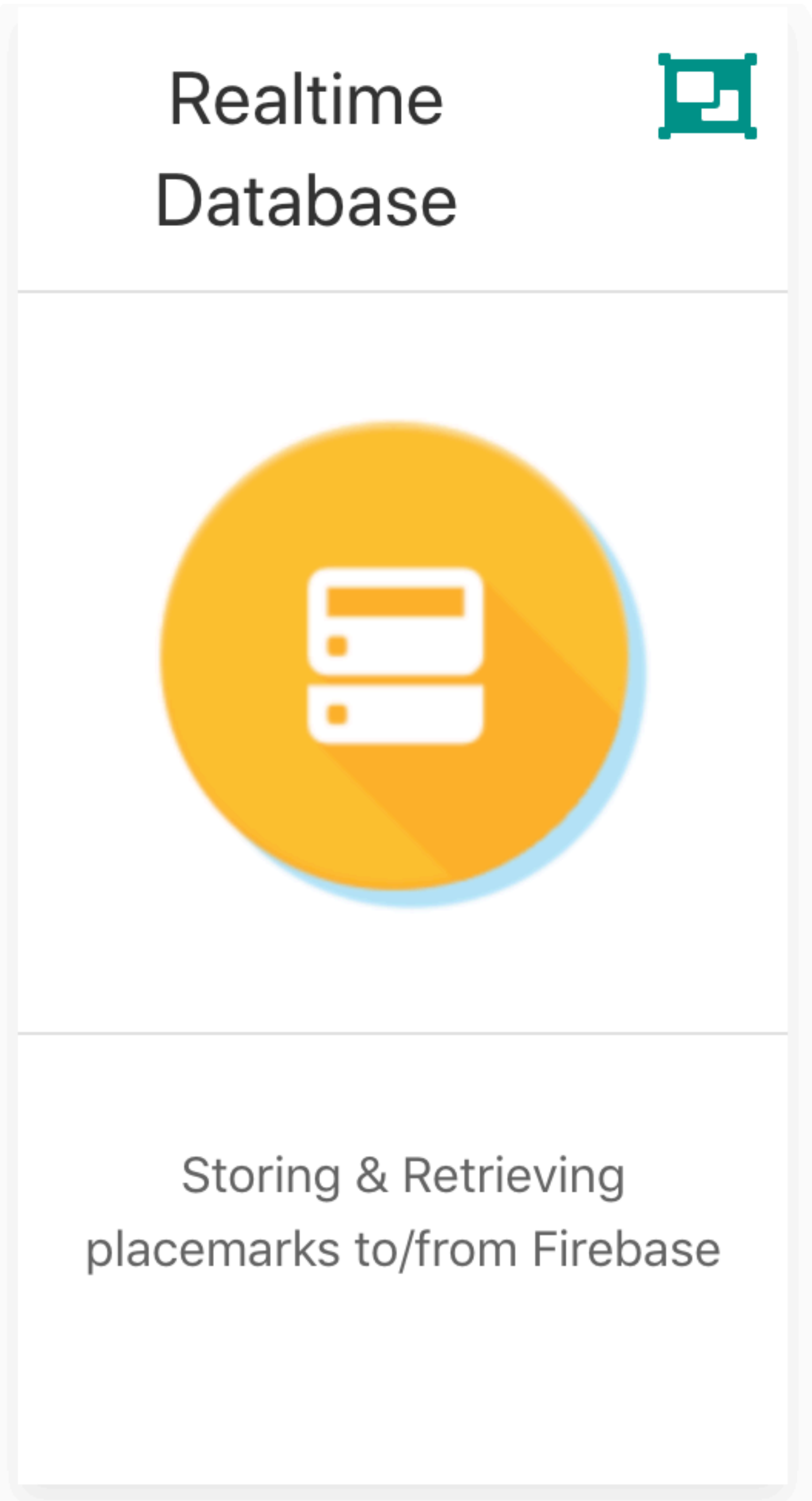
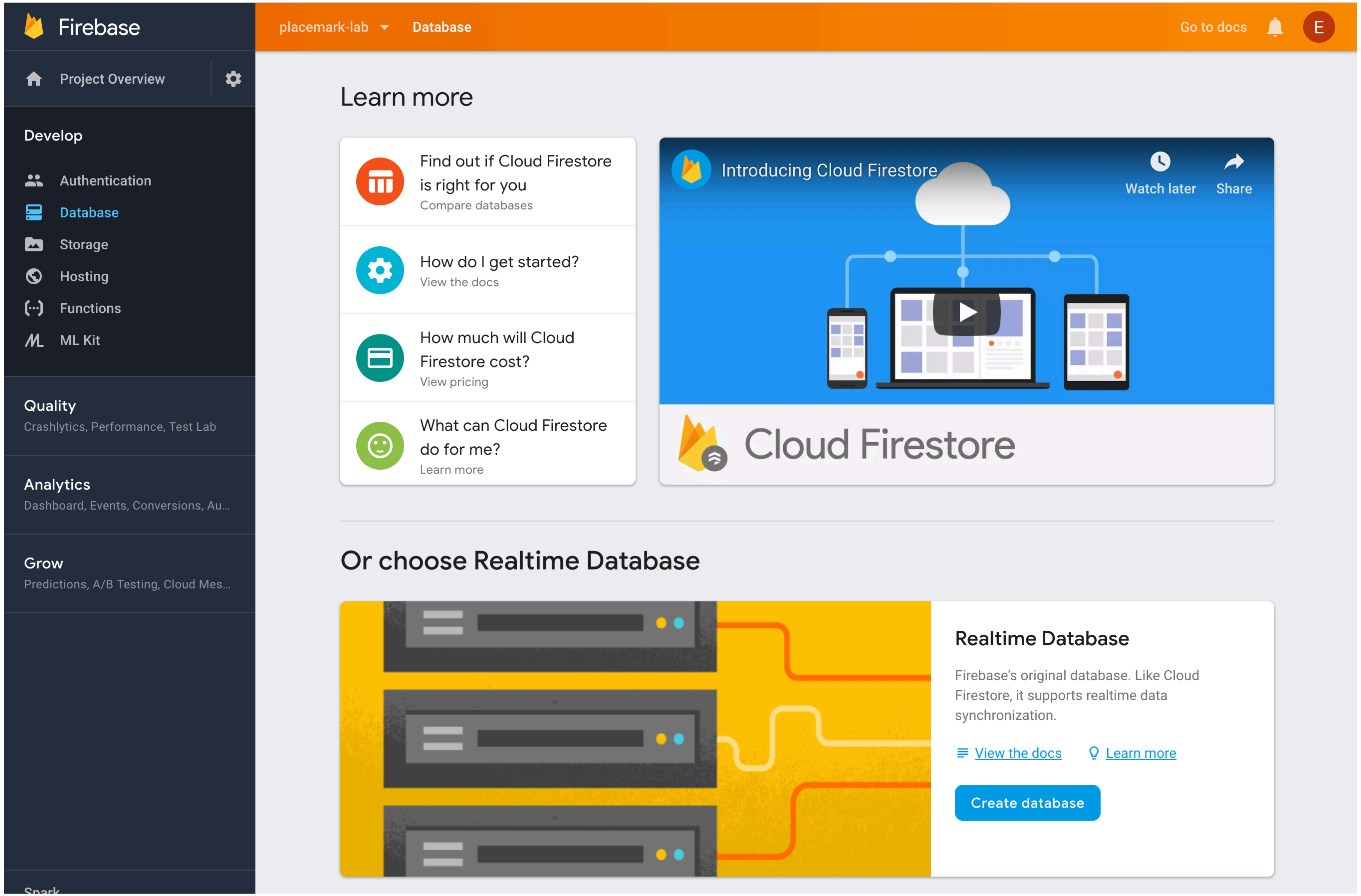
Firebase Database



Realtime Database

Storing & Retrieving placemarks to/from Firebase

Realtime Database



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

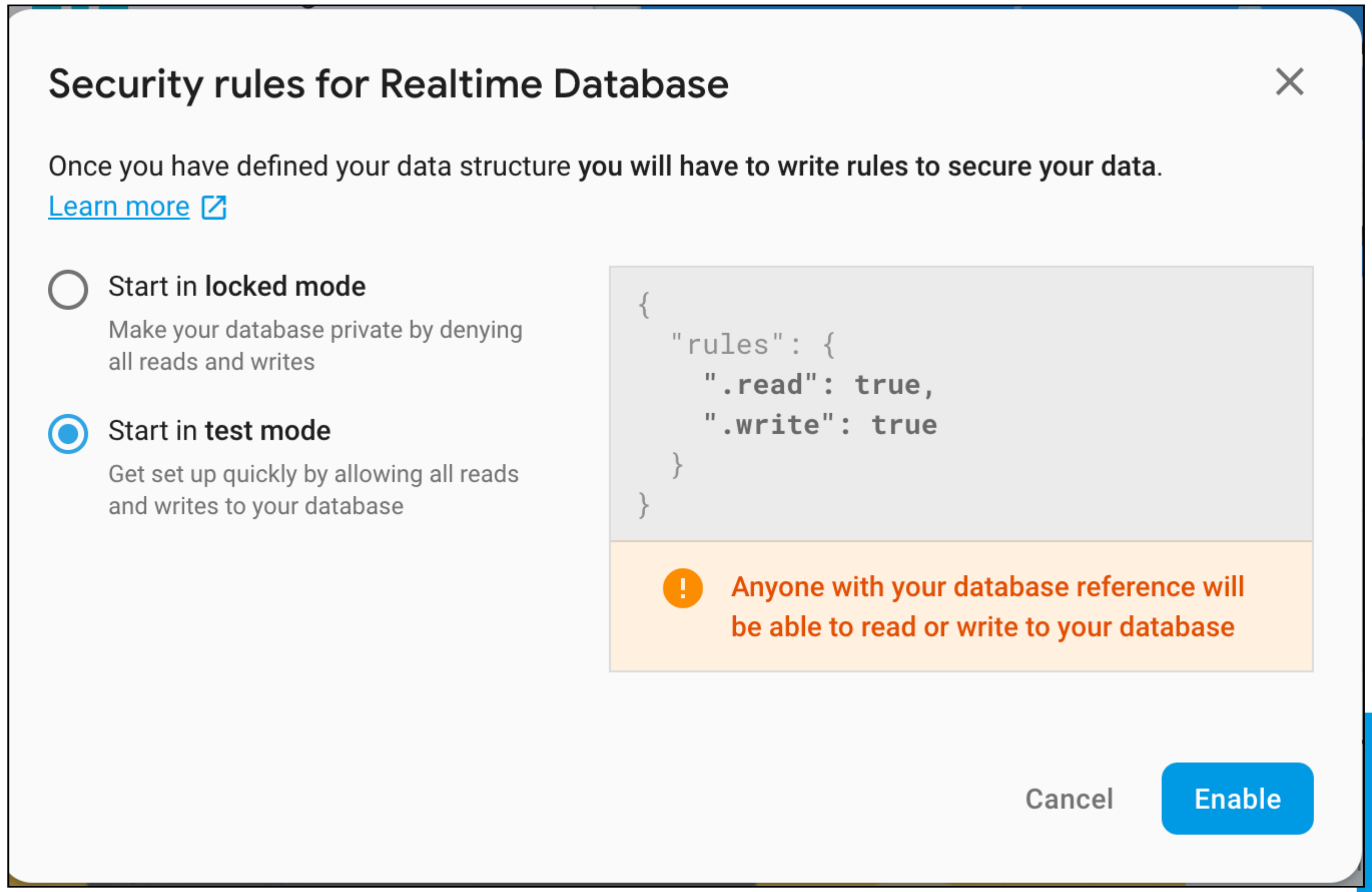
Or choose Realtime Database

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

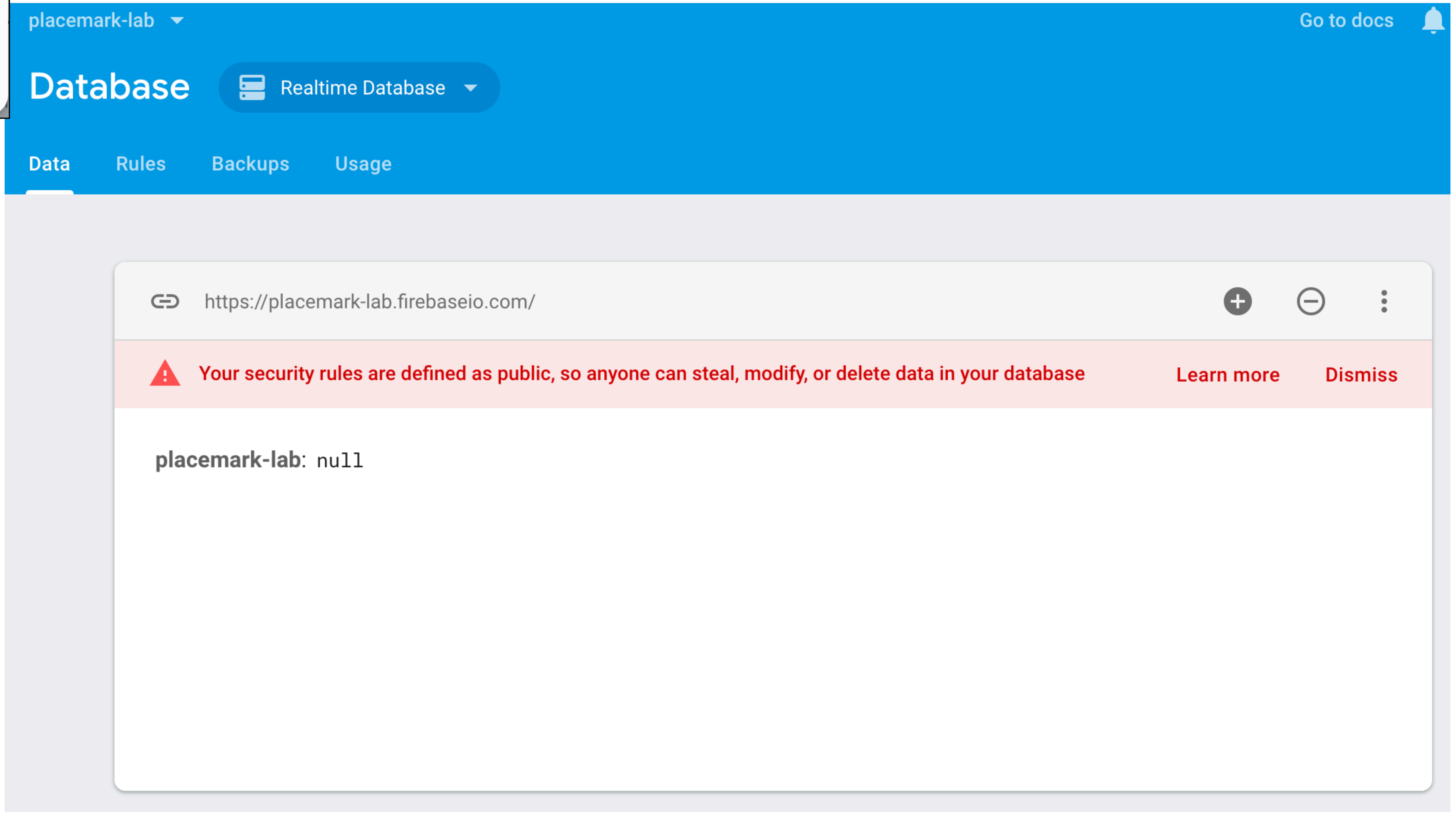
Once you have defined your data structure **you will have to write rules to secure your data**.
Learn more

- 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

https://console.firebase.google.com

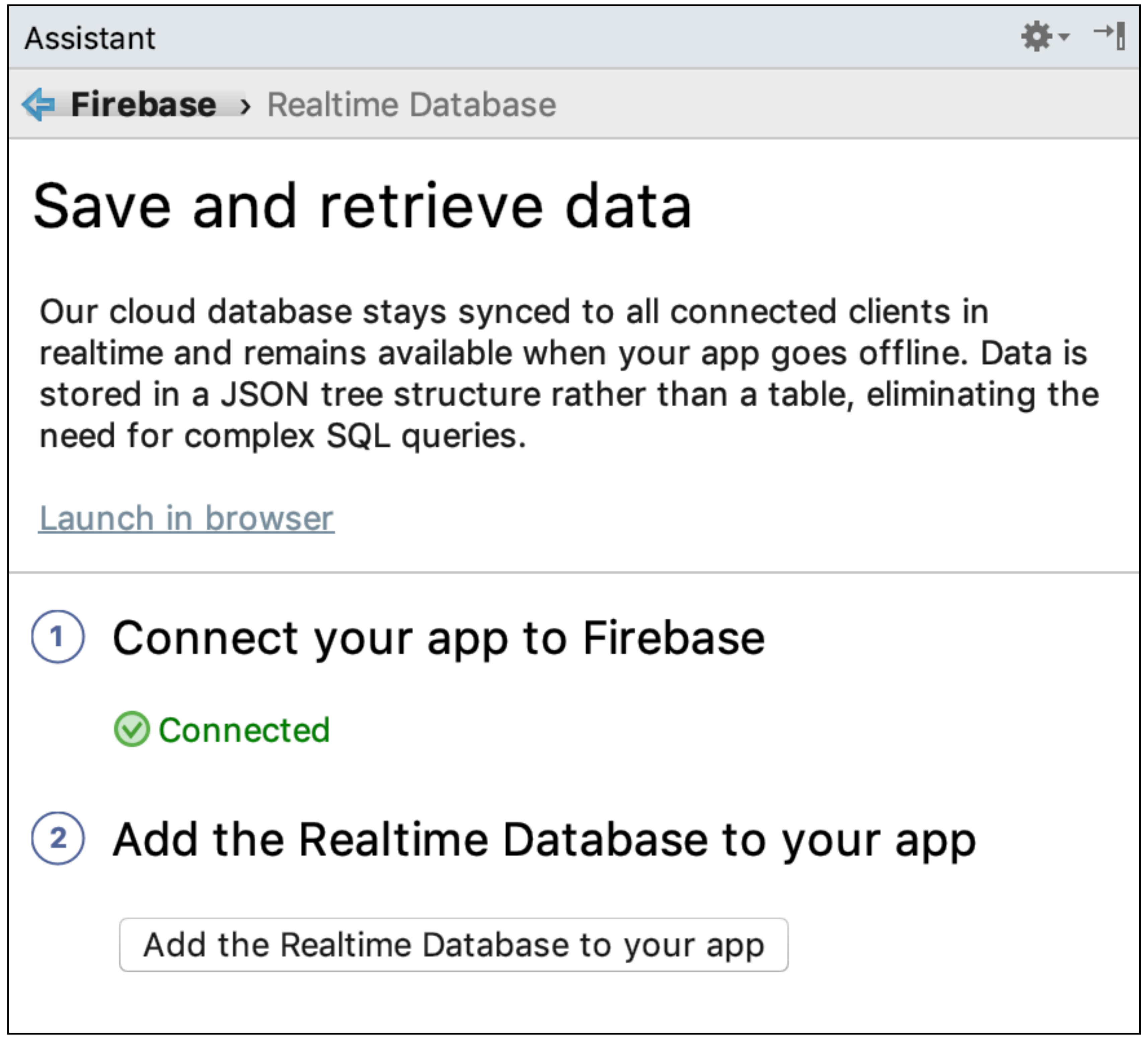


https://placemark-lab.firebaseio.com/

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

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.

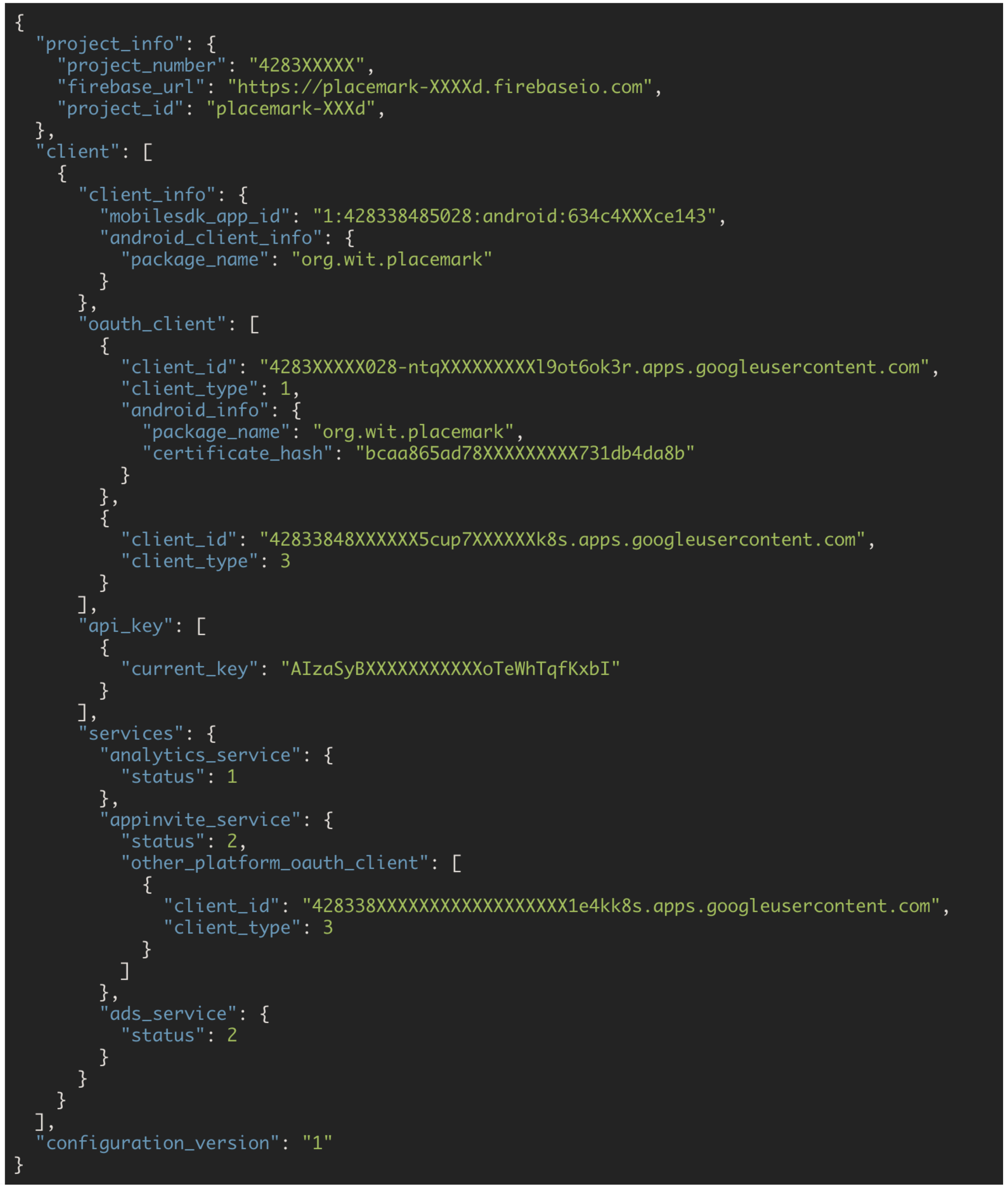
Launch in browser

1. Connect your app to Firebase

   ✓ Connected

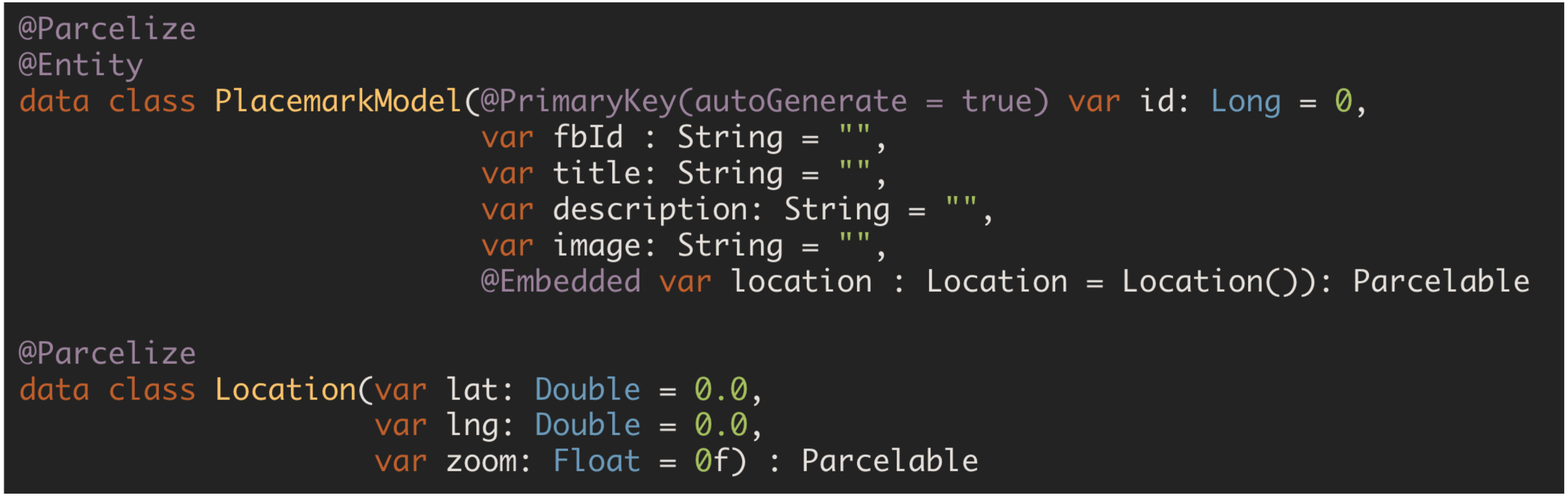
2. Add the Realtime Database to your app

   Add the Realtime Database to your app


```
{
  "project_info": {
    "project_number": "4283XXXXX",
    "firebase_url": "https://placemark-XXXXd.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": "4283XXXXX028-ntqXXXXXXXXXl9ot6ok3r.apps.googleusercontent.com",
          "client_type": 1,
          "android_info": {
            "package_name": "org.wit.placemark",
            "certificate_hash": "bcaa865ad78XXXXXXXXX731db4da8b"
          }
        },
        {
          "client_id": "42833848XXXXXX5cup7XXXXXXk8s.apps.googleusercontent.com",
          "client_type": 3
        }
      ],
      "api_key": [
        {
          "current_key": "AIzaSyBXXXXXXXXXXXoTeWhTqfKxbI"
        }
      ],
      "services": {
        "analytics_service": {
          "status": 1
        },
        "appinvite_service": {
          "status": 2,
          "other_platform_oauth_client": [
            {
              "client_id": "428338XXXXXXXXXXXXXXXXXX1e4kk8s.apps.googleusercontent.com",
              "client_type": 3
            }
          ]
        },
        "ads_service": {
          "status": 2
        }
      }
    }
  ],
  "configuration_version": "1"
}
```

Updates
app/google-services.json

PlacemarkModel

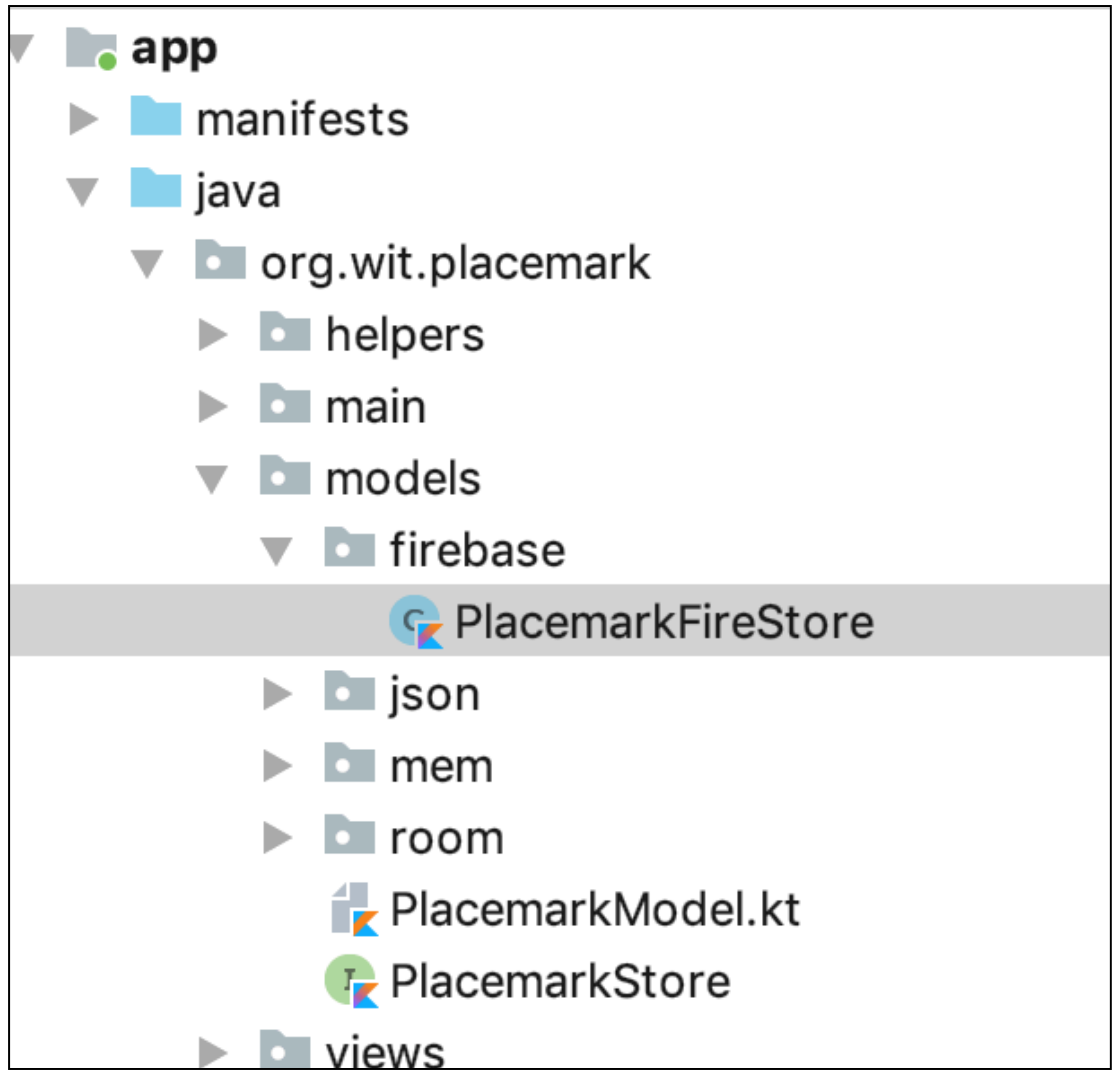


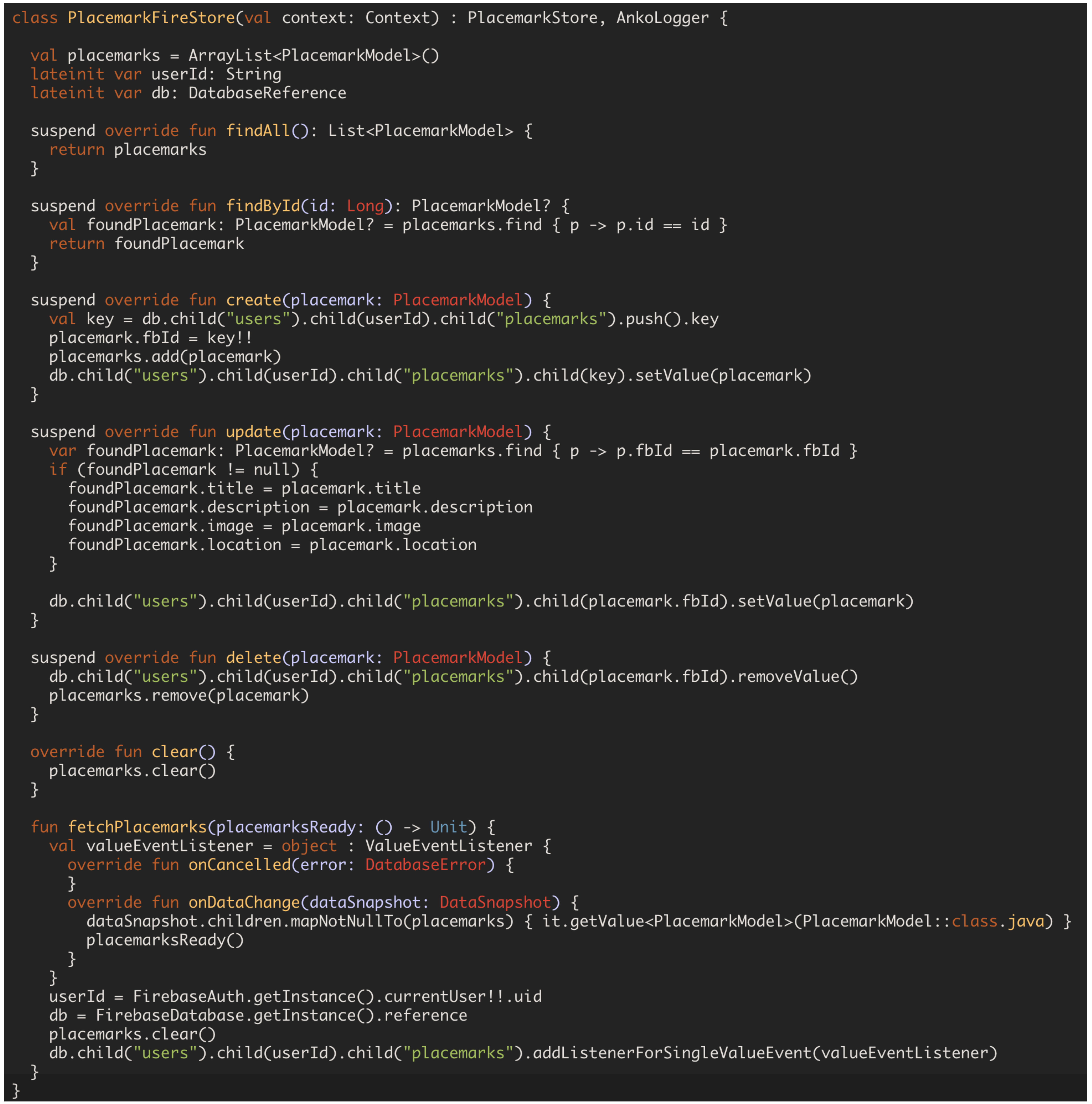
```kotlin
@Parcelize
@Entity
data class PlacemarkModel(@PrimaryKey(autoGenerate = true) var id: Long = 0,
                          var fbId : String = "",
                          var title: String = "",
                          var description: String = "",
                          var image: String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





```kotlin
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)
  }
}
```

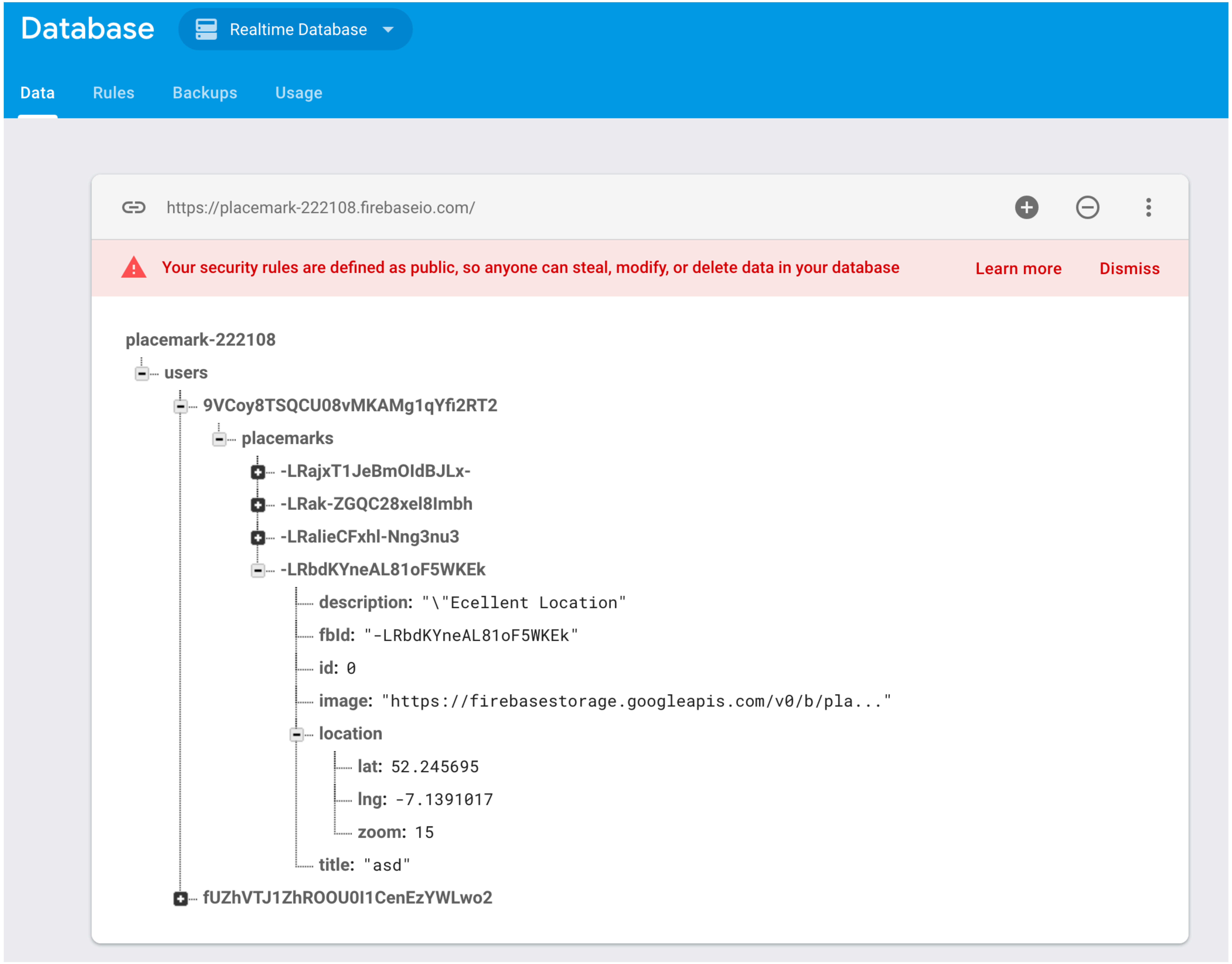
PlacemarkFireStore - Initialisation

Firebase UserID (from Auth)

Firebase Database Reference

```
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
    ...
  }
  ...
}
```

Database Structure



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

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

```
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
```

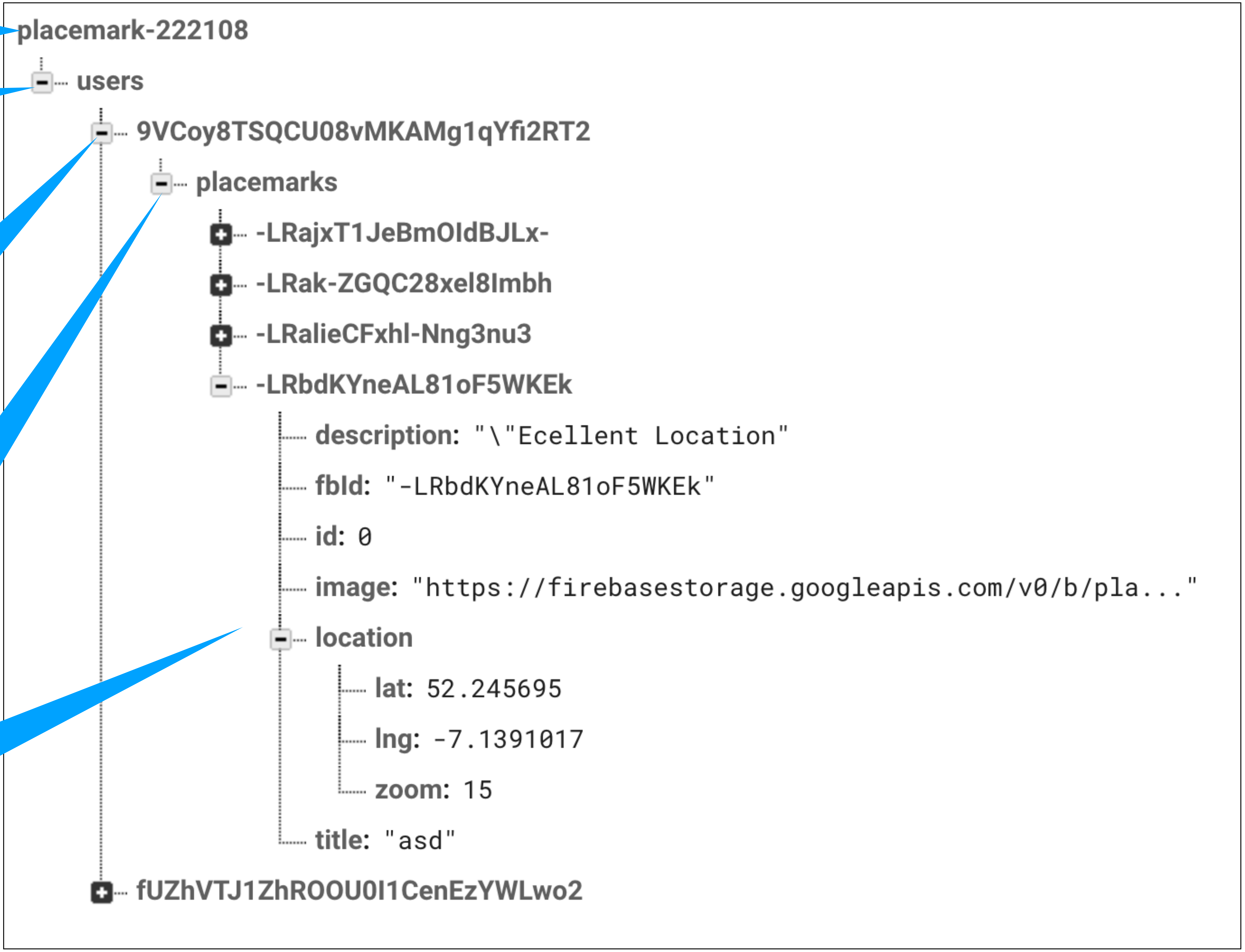
Application Database

Collection of all Users

Individual user (based in Auth ID)

This Users' placemark collection

Individual Placemark



```
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
```

PlacemarkFireStore - Create

```
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)

    }
  }

  …
}
```

Create a new Placemark object in the Database

Keep local copy of Placemark in placemarks array

Populate the object with Placemark details

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

```kotlin
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

```kotlin
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

PlacemarkFireStore - fetchPlacemarks

Lambda we will call when placemarks have been retrieved

```
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)
}
```

Trigger lambda - as place marks have been retrieved

PlacemarkFireStore - fetchPlacemarks

Lambda we will call when placemarks have been retrieved

```
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

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 : ValueEve
    override fun onCancelled(dataSnapshot: Dat
      // Error connecting to database
    }
    override fun onDataChange(dataSnapshot: Da
      dataSnapshot!!.children.mapNotNullTo(pla
        it.getValue<PlacemarkModel>(PlacemarkM
      }
      placemarksReady()
    }
  }

  userId = FirebaseAuth.getInstance().currentU
  db = FirebaseDatabase.getInstance().referenc
  placemarks.clear()

  db.child("users").child(userId).child("place
}
```

```
fun doLogin(email: String, password: String) {
  view?.showProgress()
  auth.signInWithEmailAndPassword(email, password).addOnCompleteListener(v
    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}")
    }
  }
}
```