import com.google.firebase.database.\*

// Get a reference to your Firebase database

val database = FirebaseDatabase.getInstance()

// Get a reference to the "students" collection

val studentsRef = database.getReference("students")

// Query the "students" collection and retrieve the last record

val studentsQuery = studentsRef.orderByChild("timestamp").limitToLast(1)

// Add a ValueEventListener to retrieve the data

studentsQuery.addListenerForSingleValueEvent(object : ValueEventListener {

override fun onDataChange(dataSnapshot: DataSnapshot) {

if (dataSnapshot.exists()) {

// Retrieve the last student record

val lastStudentRecord = dataSnapshot.children.last().getValue(Student::class.java)

println("Last student record: $lastStudentRecord")

} else {

println("No student data available")

}

}

override fun onCancelled(databaseError: DatabaseError) {

println("Database error: ${databaseError.message}")

}

})

// Get a reference to the "tutors" collection

val tutorsRef = database.getReference("tutors")

// Query the "tutors" collection and retrieve the last record

val tutorsQuery = tutorsRef.orderByChild("timestamp").limitToLast(1)

// Add a ValueEventListener to retrieve the data

tutorsQuery.addListenerForSingleValueEvent(object : ValueEventListener {

override fun onDataChange(dataSnapshot: DataSnapshot) {

if (dataSnapshot.exists()) {

// Retrieve the last tutor record

val lastTutorRecord = dataSnapshot.children.last().getValue(Tutor::class.java)

println("Last tutor record: $lastTutorRecord")

} else {

println("No tutor data available")

}

}

override fun onCancelled(databaseError: DatabaseError) {

println("Database error: ${databaseError.message}")

}

})

In this example:

* We have two collections: "students" and "tutors".
* We retrieve the last record from each collection separately by creating different references (**studentsRef** and **tutorsRef**) and querying each reference individually.
* Replace **Student** and **Tutor** with your actual data model classes for students and tutors, respectively.
* Ensure that the field used for sorting (**timestamp** in this case) exists in both collections if you're sorting by the same criteria. Otherwise, adjust the sorting criteria as needed for each collection.
* // Import necessary Firebase libraries
* import com.google.firebase.database.DatabaseReference
* import com.google.firebase.database.FirebaseDatabase
* import com.google.firebase.database.Query
* import com.google.firebase.database.DataSnapshot
* import com.google.firebase.database.DatabaseError
* import com.google.firebase.database.ValueEventListener
* // Function to check for duplicates and create a new record if there are none
* fun createRecordIfNotDuplicate(email: String, type: String, userData: UserModel) {
* // Get a reference to the "users" collection in the database
* val usersRef: DatabaseReference = FirebaseDatabase.getInstance().getReference("users")
* // Create a query to search for records with matching email
* val emailQuery: Query = usersRef.orderByChild("email").equalTo(email)
* // Create a query to search for records with matching type
* val typeQuery: Query = usersRef.orderByChild("type").equalTo(type)
* // Listener to check for duplicates based on email
* emailQuery.addListenerForSingleValueEvent(object : ValueEventListener {
* override fun onDataChange(dataSnapshot: DataSnapshot) {
* if (dataSnapshot.exists()) {
* // If there are matching records for the email
* println("Duplicate record found based on email. Record not created.")
* } else {
* // Proceed to check type duplication
* checkTypeDuplicate(typeQuery, userData)
* }
* }
* override fun onCancelled(databaseError: DatabaseError) {
* // Handle database error
* println("Database error: ${databaseError.message}")
* }
* })
* }
* // Function to check for duplicates based on type
* fun checkTypeDuplicate(typeQuery: Query, userData: UserModel) {
* // Listener to check for duplicates based on type
* typeQuery.addListenerForSingleValueEvent(object : ValueEventListener {
* override fun onDataChange(dataSnapshot: DataSnapshot) {
* if (dataSnapshot.exists()) {
* // If there are matching records for the type
* println("Duplicate record found based on type. Record not created.")
* } else {
* // No duplicates found, create the new record
* createNewRecord(userData)
* }
* }
* override fun onCancelled(databaseError: DatabaseError) {
* // Handle database error
* println("Database error: ${databaseError.message}")
* }
* })
* }
* // Function to create a new record in the database
* fun createNewRecord(userData: UserModel) {
* // Get a reference to the "users" collection in the database
* val usersRef: DatabaseReference = FirebaseDatabase.getInstance().getReference("users")
* // Generate a new key for the new record
* val newRecordKey = usersRef.push().key
* // Set the user data at the new record key
* if (newRecordKey != null) {
* usersRef.child(newRecordKey).setValue(userData)
* .addOnSuccessListener {
* // Success handling
* println("New record created successfully.")
* }
* .addOnFailureListener { error ->
* // Failure handling
* println("Failed to create new record: ${error.message}")
* }
* }
* }
* // Call the function to create a new record if there are no duplicates
* createRecordIfNotDuplicate(email = "example@example.com", type = "student", userData = userModel)Top of Form