Ei Androider, já ouviu “a palavra” do Firebase?

Quem disse que nossa aplicação Android precisa de um backend muito Robusto?

inter

Daivid Vasconcelos Leal
Mestrando na UFRPE
Sr. Android Developer no Inter

daivid.v.leal

+55 81 9 9525 2247

Agenda

1. Inicializando o Firebase
2. Cloud Firestore
3. Realtime Database
4. Android Storage

Inicializando o Firebase

Os tópicos abaixo descrevem os passos para criar uma aplicação no Firebase.

Projeto

Devemos primeiro criar um projeto e definir um nome e dizer se utilizaremos ou não o Analytics no projeto.

Adicionar o app no Firebase

Definiremos em seguida alguns passos para adicionar o projeto no Firebase.

Inicializar o Firebase no App

Como inicializar o projeto no Firebase, utilizando apenas uma linha de código.

Inicializando o Firebase

/ **Projeto**

/ Adicionar o Firebase

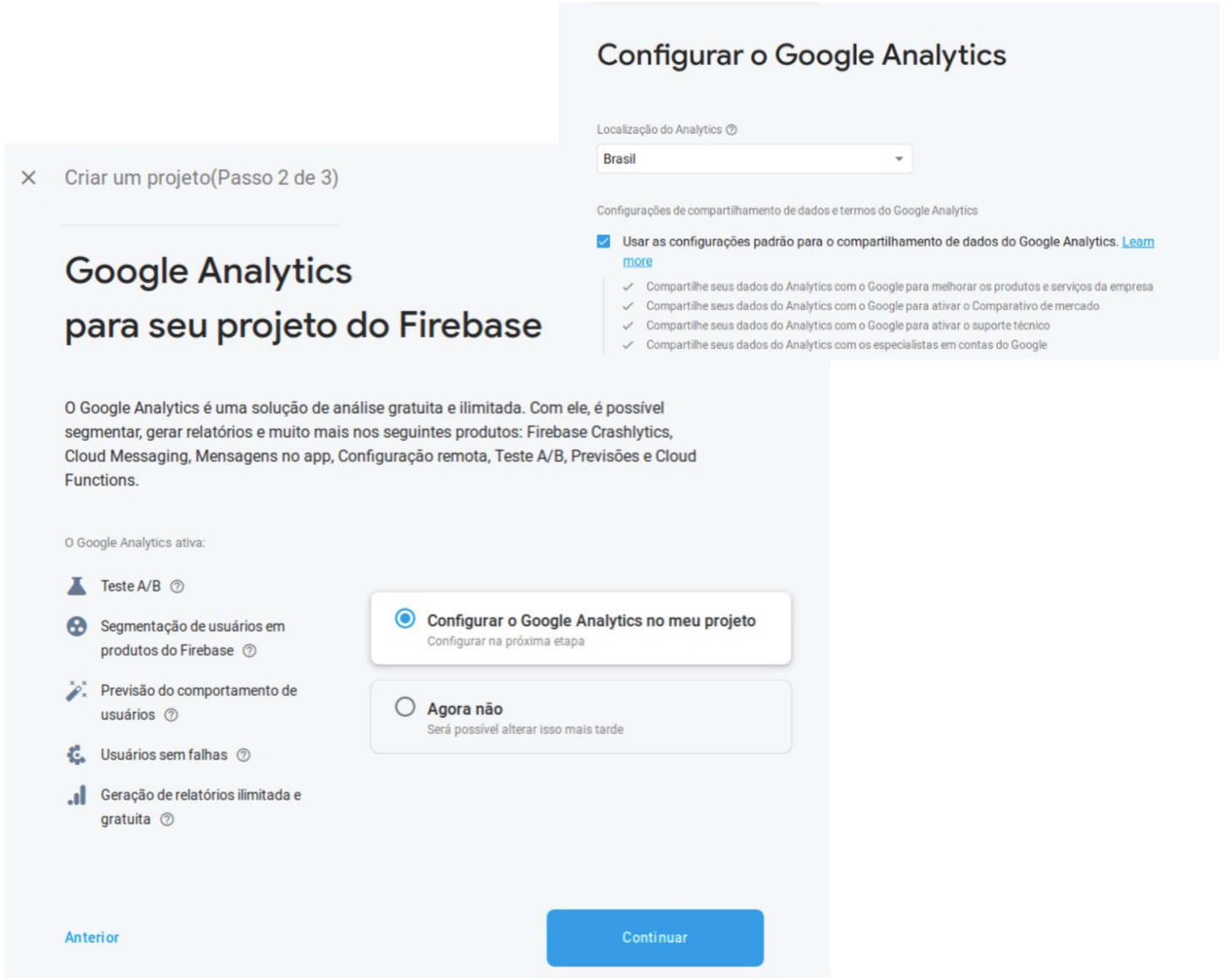
/ Inicializar o Firebase

Inicializando o Firebase

/ **Projeto**

/ Adicionar o Firebase

/ Inicializar o Firebase



Inicializando o Firebase

/ Projeto

/ **Adicionar o Firebase**

/ Inicializar o Firebase

Linux/ Mac OS:

> *keytool -list -v \-alias androiddebugkey -keystore ~/.android/debug.keystore*

Windows:

> *keytool -list -v \-alias androiddebugkey -keystore %USERPROFILE%\.android\debug.keystore*

Inicializando o Firebase

/ Projeto

/ **Adicionar o Firebase**

/ Inicializar o Firebase

Projeto Android

Com o Firebase configurado, damos início ao desenvolvimento do projeto. Vamos tomar como base um projeto que esteja utilizando Koin para injetar as dependências necessárias e MVVM como arquitetura.

No build.gradle do módulo app, adicione ao final do arquivo a seguinte linha de código:

apply plugin: 'com.google.gms.google-services'

Isso vai possibilitar a leitura do arquivo .json gerado pelo Firebase que você deve adicionar no módulo app, além de aplicar algumas dependências que são necessárias para rodar os serviços solicitados ao Firebase.

Adicione também a dependência core do Firebase.

Implementation "com.google.Firebase:Firebase-core:17.2.1"

Inicializando o Firebase

/ Projeto

/ Adicionar o Firebase

/ **Inicializar o Firebase**

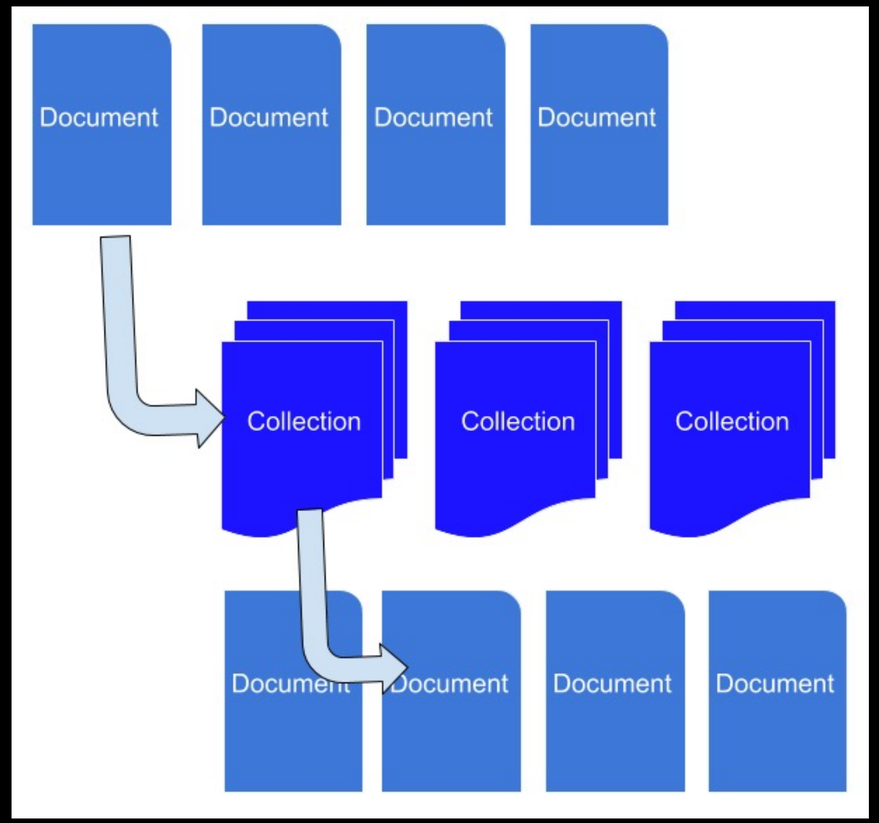
```
import android.app.Application
import com.google.firebase.FirebaseApp
import org.koin.android.ext.koin.androidContext
import org.koin.core.context.startKoin

class MyApplication : Application(){

    override fun onCreate() {
        super.onCreate()
        FirebaseApp.initializeApp(this)
        startKoin {
            androidContext(this@MyApplication)
            modules(listOf(
                networkModule,
                repositoryModule,
                viewModelModule
            ))
            properties(
                 mapOf(PROPERTY_BASE_URL to BuildConfig.BASE_URL))
        }
    }
}
```

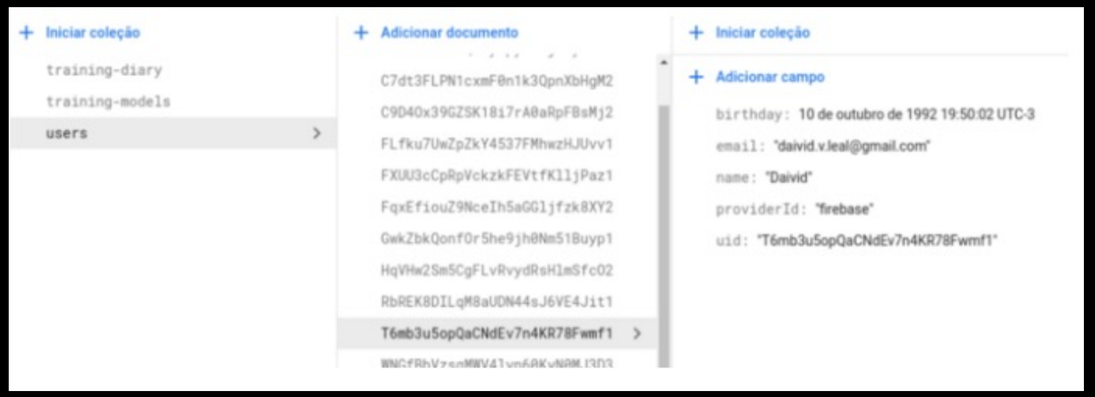
O Firebase nasceu que nem os #sanguelaranjas, nasceu pra fazer diferente.

Cloud Firestore

Possibilita consultas eficientes, atualizações e escalonamento automático de dados.

Ex: armazenamento de dados de usuários.





Cloud Firestore

/ **Um pouco de teoria**

/ Primeiros passos

/ Tratando dados

Projeto Android

Com o Firebase já inicializado, temos que adicionar a seguinte biblioteca:

Implementation “com.google.firebase:firebase-firestore:21.2.1”

Para recuperar a instância da base dados utilizamos a função *FirebaseFirestore.getInstanve( ).*

Cloud Firestore

/ Um pouco de teoria

/ **Primeiros passos**

/ Tratando dados

Projeto Android

Precisamos informar a instância que solicitamos do Firestore uma regra para habilitar a persistência dos dados.

Sendo assim, temos:

```
    single {
        val fireStore = FirebaseFirestore.getInstance()
        fireStore.firestoreSettings =
FirebaseFirestoreSettings.Builder()
            .setPersistenceEnabled(true)
            .build()
        fireStore
    }
```

Cloud Firestore

/ Um pouco de teoria

/ **Primeiros passos**

/ Tratando dados

Salvando dados de um Usuário

```
override fun signUp(
    name: String,
    birthday: Timestamp,
    email: String,
    password: String,
    success: (FirebaseUser?) -> Unit,
    error: (Exception?) -> Unit
) {
    auth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val id = auth.currentUser?.uid?.let {
                    it
                } ?: run { email }
                database.collection(COLLECTION_USER).document(id).set(
                    User(
                        name = name,
                        birthday = birthday,
                        email = email,
                        providerId = auth.currentUser?.providerId,
                        uId = auth.currentUser?.uid
                    )
                ).addOnSuccessListener {
                    success(auth.currentUser)
                }.addOnFailureListener{
                    error(it)
                }
            } else {
                error(task.exception)
            }
        }
}
```

Cloud Firestore

/ Um pouco de teoria

/ Primeiros passos

/ **Tratando dados**

Recuperando dados de um Usuário

```
database.collection(COLLECTION_USER).document(id).set(
                    User(
                        name = name,
                        birthday = birthday,
                        email = email,
                        providerId = auth.currentUser?.providerId,
                        uId = auth.currentUser?.uid
                    )
                ).addOnSuccessListener {
                    success(auth.currentUser)
                }.addOnFailureListener{
                    error(it)
                }
```

```
database.collection(COLLECTION_USER).document(uid).get().addOnSucces
sListener {
    success(it.toObject(User::class.java))
}.addOnFailureListener {
    error(it)
}
```

Cloud Firestore

/ Um pouco de teoria

/ Primeiros passos

/ **Tratando dados**

Inserindo dados de um treinamento

```
override fun insertTrainingDiary(
    date: String,
    trainingDiary: TrainingDiary,
    success: () -> Unit,
    error: (Exception?) -> Unit
) {
    auth.currentUser?.uid?.let { uid ->
        database
            .collection(COLLECTION_TRAINING_DIARY)
            .document(uid)
            .collection(COLLECTION_DIARY)
            .document(date).set(trainingDiary)
            .addOnSuccessListener {
                success()
            }
            .addOnFailureListener {
                error(it)
            }
    } ?: run {
        error(Exception())
    }
}
```

Cloud Firestore

/ Um pouco de teoria

/ Primeiros passos

/ **Tratando dados**

Recuperando um treinamento

```
database
    .collection(COLLECTION_TRAINING_DIARY)
    .document(uid)
    .collection(COLLECTION_DIARY)
    .document(date)
    .get()
    .addOnSuccessListener {
        success(it.toObject(TrainingDiary::class.java))
    }
    .addOnFailureListener {
        error(it)
    }
```

Recuperando uma collection de treinamentos

```
database
    .collection(COLLECTION_TRAINING_DIARY)
    .document(uid)
    .collection(COLLECTION_DIARY).get()
    .addOnSuccessListener {
        success(it.toObjects(TrainingDiary::class.java))
    }
    .addOnFailureListener {
        error(it)
    }
```

Cloud Firestore

/ Um pouco de teoria

/ Primeiros passos

/ **Tratando dados**

Aplicando Filtros

```
database
    .collection(COLLECTION_TRAINING_DIARY)
    .document(uid)
    .collection(COLLECTION_DIARY)
    .whereGreaterThanOrEqualTo(FieldPath.documentId(), startDate)
    .whereLessThanOrEqualTo(FieldPath.documentId(), endDate)
    .get()
    .addOnSuccessListener {
        success(it.toObjects(TrainingDiary::class.java))
    }
    .addOnFailureListener {
        error(it)
    }
```

Cloud Firestore

/ Um pouco de teoria

/ Primeiros passos

/ **Tratando dados**

Deletando Conteúdo

```
override fun deleteTrainingDiary(
    date: String,
    trainingDiary: TrainingDiary,
    success: () -> Unit,
    error: (Exception?) -> Unit
) {
    auth.currentUser?.uid?.let { uid ->
        database.collection(COLLECTION_TRAINING_DIARY)
        .document(uid)
        .collection(COLLECTION_DIARY)
        .document(date).delete()
        .addOnSuccessListener {
            success()
        }
        .addOnFailureListener {
            error(it)
        }
    } ?: run {
        error(Exception())
    }
}
```

Cloud Firestore

/ Um pouco de teoria

/ Primeiros passos

/ **Tratando dados**

Realtime Database

Um Database com atualização em tempo real.

É um banco de dados que também utiliza um modelo não estruturado. O NoSQL no Firebase está presente em quase todas os tipos de dados.

Realtime Database

Com dois passos é fácil manter sua tela atualizada com dados remotos e em tempo real.

Regras

Regras de acesso são necessárias para definir que tipo de usuário pode ou não acessar seus os dados.

Crie um Listener

Mostraremos como podemos criar um mecanismo para atualizar os dados na tela.

Realtime Database

/ **Primeiros passos**

/ Regras

/ Listener

/ Tratando dados

Projeto Android

Com o Firebase já inicializado, temos que adicionar mais a seguinte biblioteca:

Implementation “com.google.firebase:firebase-database:19.2.0”

Para recuperar a instância da base dados utilizamos a função *FirebaseDatabase.getInstance( ).*

Realtime Database

/ Primeiros passos

/ **Regras**

/ Listener

/ Tratando dados

```
// Esta regra IMPOSSIBILITA PARA QUALQUER USUÁRIO a leitura e
escrita
 // na nossa base de dados
{
  "rules": {
    ".read": false,
    ".write": false
  }
}


// Esta regra POSSIBILITA PARA QUALQUER USUÁRIO a leitura e escrita
 // na nossa base de dados
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

Realtime Database

/ Primeiros passos

/ Regras

/ **Listener**

/ Tratando dados

```
class ServicePost(private val realtimeDatabase: FirebaseDatabase,
private val auth: FirebaseAuth): ServicePostContract{

    private val POSTS_REF = "posts"

    override fun getReference(): DatabaseReference =
realtimeDatabase.getReference(POSTS_REF)

    override fun insertPost(post: Post) {
        ...
    }

    override fun deletePost(post: Post) {
        ...
    }
}
```

Realtime Database

/ Primeiros passos

/ Regras

/ **Listener**

/ Tratando dados

```
class PostViewModel(private val servicePostContract:
ServicePostContract): ViewModel(),
    KoinComponent {

    val post = MutableLiveData<BaseModel<List<Post>, Throwable>>()

    fun setUpListener(){
        val postListener: ValueEventListener = object :
ValueEventListener {
            override fun onDataChange(dataSnapshot: DataSnapshot) {
                val postList = ArrayList<Post>()
                for (postSnapshot in dataSnapshot.children) {
                    val newPost =
postSnapshot.getValue(Post::class.java)
                    newPost?.let{
                        postList.add(it)
                    }
                }
                post.value = BaseModel(BaseModel.Status.SUCCESS,
postList, null)
            }

            override fun onCancelled(databaseError: DatabaseError) {
                post.value = BaseModel(BaseModel.Status.ERROR, null,
Throwable("We try to load a new Post but it failed!"))
                Log.w("Load Post: ", "loadPost:onCancelled",
databaseError.toException())
            }
        }

servicePostContract.getReference().addValueEventListener(postListene
r)
    }
}
```

Realtime Database

/ Primeiros passos

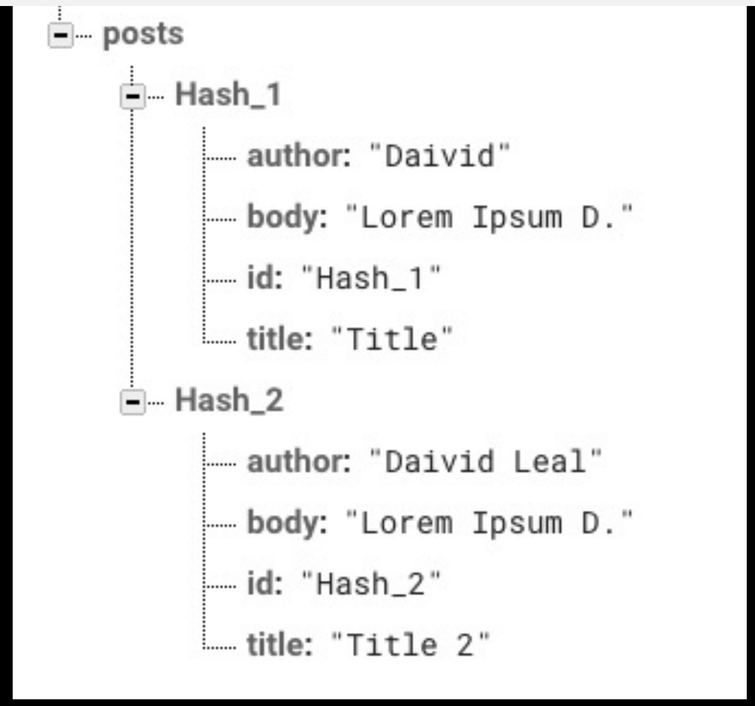
/ Regras

/ Listener

/ **Tratando dados**

```
override fun insertPost(post: Post) {
    val key: String? = getReference().push().getKey()
    post.id = key
    val postValues = post.toMap()
    val childUpdates: MutableMap<String, Any> = HashMap()
    childUpdates["/$POSTS_REF/$key"] = postValues
    realtimeDatabase.getReference().updateChildren(childUpdates)
        .addOnSuccessListener{}.addOnFailureListener{}
}
```

```
override fun deletePost(post: Post) {
    getReference().child(post.id ?: "").removeValue()
        .addOnFailureListener { }
        .addOnSuccessListener { }
}
```



Criar, simplificar,

resolver, surpreender.

Android Storage

E para que usamos o **Storage?** Basicamente, para armazenamento de arquivos, sejam esses arquivos imagens, vídeos e áudio.

Realtime Database

/ **Primeiros passos**

/ Regras

/ Tratando Arquivos

Projeto Android

Com o Firebase já inicializado, temos que adicionar mais a seguinte biblioteca:

Implementation “com.google.firebase:firebase-storage:19.1.0”

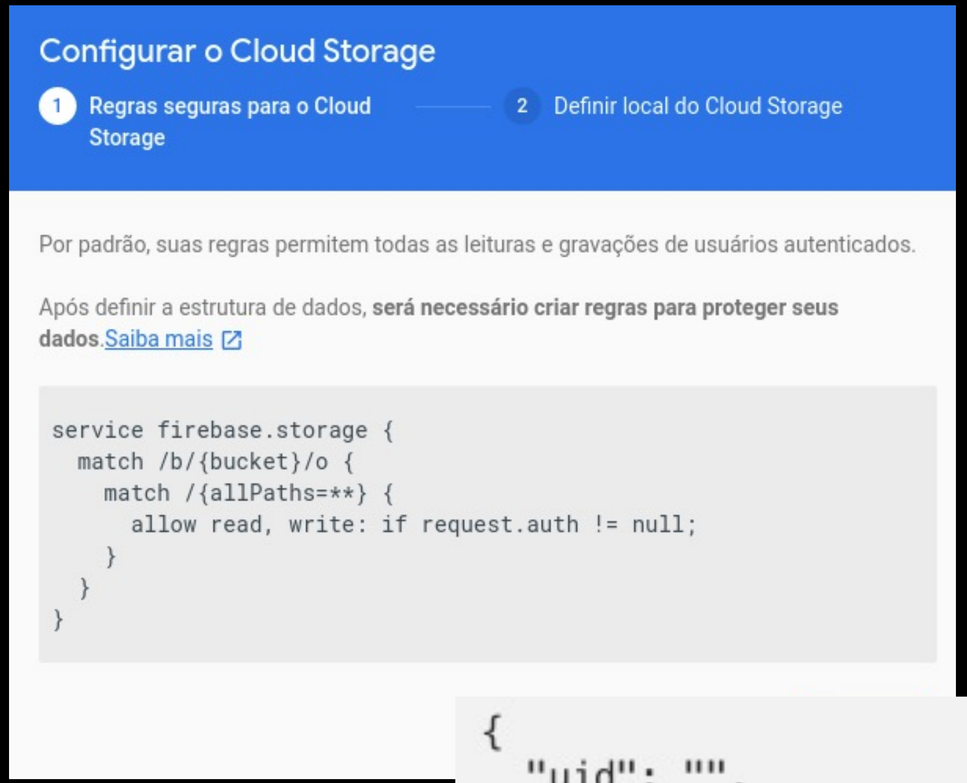
Com essa biblioteca adicionada teremos acesso ao método *FirebaseStorage.getInstance( )* para poder utilizar as funções disponíveis de adição e remoção de arquivos.

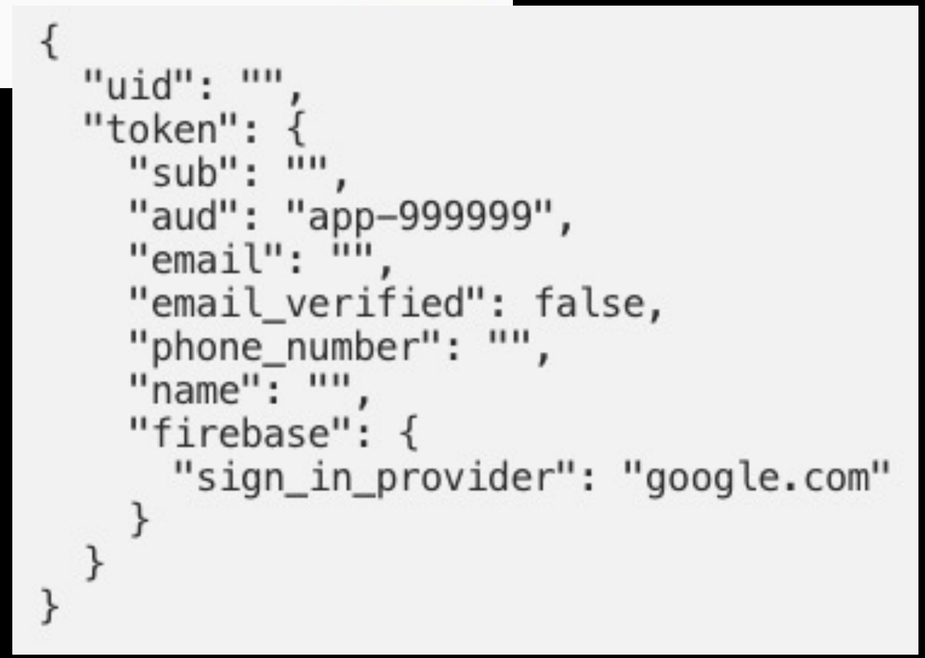
Realtime Database

/ Primeiros passos

/ **Regras**

/ Tratando Arquivos





Realtime Database

/ Primeiros passos

/ Regras

/ **Tratando Arquivos**

```
fun uploadPhoto(
    uriFile: Uri,
    fileName: String,
    success: () -> Unit,
    error: (Exception?) -> Unit
) {
    val ref = firebaseStorage.getReference(fileName)
    ref.putFile(uriFile).addOnCompleteListener{
        success()
    }.addOnFailureListener{
        error(it)
    }
}
```

```
fun deleteFile(
    fileDir: String,
    success: (task: Task<Void>) -> Unit,
    error: (java.lang.Exception?) -> Unit
) {
    val ref = firebaseStorage.reference.child(fileDir)
    ref.delete().addOnFailureListener(error)
                .addOnCompleteListener(success)
}
```

Realtime Database

/ Primeiros passos

/ Regras

/ **Tratando Arquivos**

```
fun uploadPhoto(
    uriFile: Uri,
    fileName: String,
    success: (uri: Uri) -> Unit,
    error: (Exception?) -> Unit
) {
    val ref = firebaseStorage.getReference(fileName)
    val task = ref.putFile(uriFile)
    this.generateUrlDownload(ref, task, success, error)
}

private fun generateUrlDownload(
    reference: StorageReference,
    task: StorageTask<UploadTask.TaskSnapshot>,
    success: (uri: Uri) -> Unit,
    error: (Exception?) -> Unit
) {
    task.continueWithTask { taskExecuted ->
        if(taskExecuted.isSuccessful) {
            reference.downloadUrl
        } else {
            taskExecuted.exception?.let {
                throw it
            }
        }
    }.addOnCompleteListener { task ->
        if (task.isSuccessful) {
            task.result?.let(success) ?: run {
                error(Throwable("Unknown Error"))
            }
        } else {
            error(Throwable("Unknown Error!"))
        }
    }.addOnFailureListener(error)
}
```

Realtime Database

/ Primeiros passos

/ Regras

/ **Tratando Arquivos**

```
fun downloadFile(
    fileName: String,
    fileType: String,
    fileExt: String,
    success: (file: File) -> Unit,
    error: (Exception?) -> Unit
) {
    val ref = firebaseStorage.reference.child(fileName)
    val saveFileInto = File.createTempFile(fileType, fileExt)
    ref.getFile(saveFileInto).addOnSuccessListener {
        success(saveFileInto)
    }.addOnFailureListener(error)
}
```

Dúvidas?

inter