

# Apunts

Descarregar aquests apunts en [pdf](#) y [html](#)

## Tema 11.3. Persistència de Dades III Firebase

### Índex

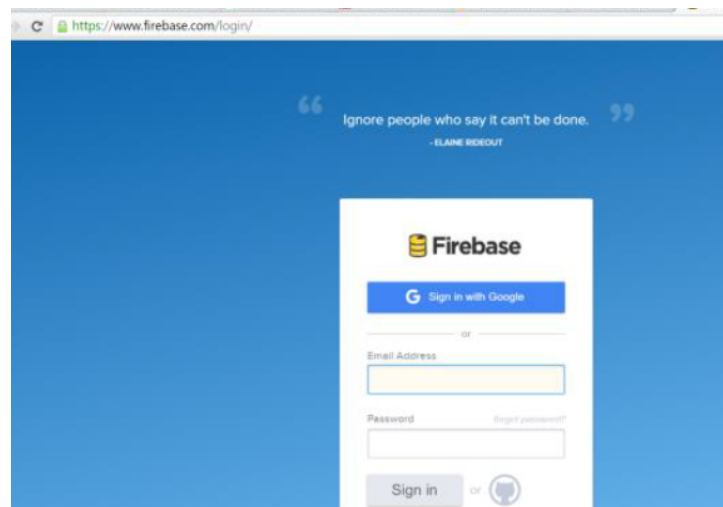
1. [Acceso a Base de Datos con FIREBASE](#)
  1. [Creant un app de Firebase](#)
  2. [Firestore DataBase](#)
    1. [FirebaseUI y RecyclerView](#)
    2. [Filtrat i ordenació](#)
  3. [Firebase Auth. Autenticació](#)

# Acceso a Base de Datos con FIREBASE

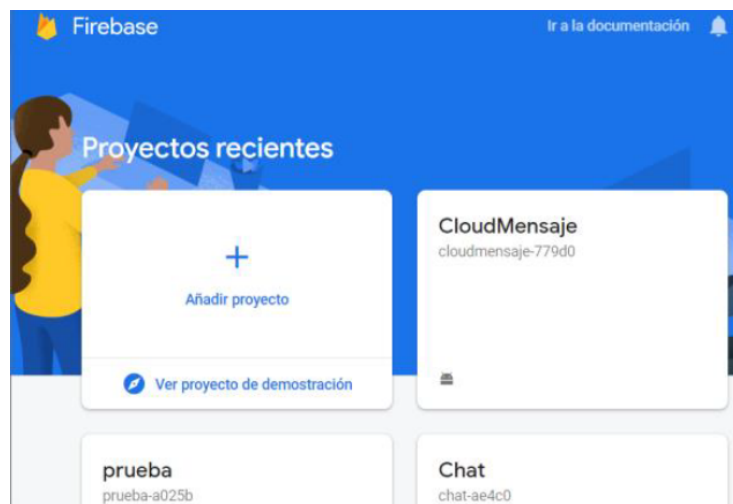
**Firebase** és una plataforma de \*backend per a construir aplicacions mòbils i web, s'encarrega del maneig de la infraestructura permetent que el desenvolupador s'enfoque en altres aspectes de l'aplicació. Entre les seues característiques s'inclou base de dades de temps real, autenticació d'usuaris i emmagatzematge (\*hosting) estàtic. La idea d'usar Firebase és no haver d'escriure codi del costat del servidor i aprofitar al màxim les característiques que ens proveeix la plataforma. Per a utilitzar Firebase amb Android disposem d'un SDK, el qual cosa ens permetrà integrar-ho fàcilment a la nostra aplicació. Per a aquest exemple construirem un llistat de coses per fer usant la base de dades de temps real de Firebase com backend i autenticació amb email/password.

## Creant un app de Firebase

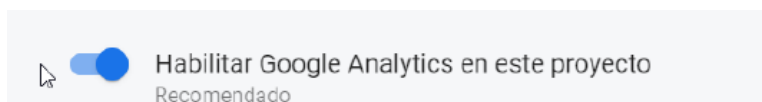
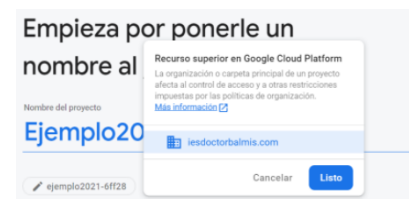
El primer pas és associar el nostre compte de Firebase a una dels nostres comptes de correu electrònic:



Una vegada està llesta veurem la pantalla on és possible visualitzar els nostres projectes o crear nous.



Per a crear un nou projecte solament haurem d'introduir un nom i l'organització o carpeta principal del projecte. Google ens demanarà si volem habilitar Google Analytics en el projecte, el deixarem habilitat.



I ho associarem al compte per defecte. Una vegada realitzats aquests passos, es crearà el projecte de firebase.

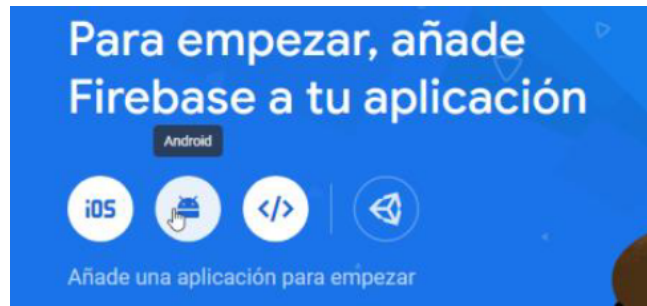
Posteriorment ens apareixerà la pantalla que ens permetrà afegir l'aplicació al nostre projecte. Cada projecte estarà associat a una o més aplicacions, per la qual cosa abans de crear el projecte necessitarem crear l'aplicació o decidir l'espai de noms d'aquesta.

*Crearem una aplicació senzilla amb el següent aspecte, que ens permeti entendre el funcionament d'accés a Firebase per a enviar i rebre dades sense autenticació.*



Quan l'usuari meta la informació en els TextView i preme el botó afegir, la informació es guarda en Firebase i quan es modifiqui aquesta des de Firebase, es mostrarà un

missatge amb el text modificat en el Txt de l'aplicació.



Una vegada tinguem l'espai de nom, podrem seguir amb els passos que ens demana Firebase per a crear l'aplicació. En el nostre cas no serà necessari introduir la signatura de certificació.

A screenshot of a web form titled "Añadir Firebase a una aplicación de Android". It has a close button (X) in the top left. The form is divided into sections by a vertical line. The first section is labeled "1 Registrar la aplicación". It contains three input fields: "Nombre del paquete de Android" with the value "com.tema12.cloudfirestore.ejemplousuarios", "Apodo de la aplicación (opcional)" with the value "EjemploUsuariosCloudFirestore20", and "Certificado de firma de depuración SHA-1 (opcional)". Below the last field is a note: "Obligatoria para Dynamic Links, Firebase Invites y la asistencia por teléfono o el inicio de sesión de Google en Auth. Edita los SHA-1 en la configuración." At the bottom is a blue button labeled "Siguiente".

Just en el moment que acaba de crear-se l'aplicació, es descarregara un arxiu JSON que haurem de copiar en el nostre projecte Android. Només haurem de seguir les instruccions que proporciona molt clarament la pàgina Web de Firebase (copiar l'arxiu i afegir les línies que indiquen que usarem serveis de google en els build.gradle de l'app i del projecte).

#### 1. En el projecte

```
dependencies {  
    classpath "com.android.tools.build:gradle:7.0.3"  
    classpath "com.google.gms:google-services:4.3.10" //añadir esta línea  
    ...  
}
```

#### 2. En la APP

```

plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'com.google.gms.google-services' // Google Services plugin
}

dependencies {
    ...
    implementation 'androidx.appcompat:appcompat:1.3.1'
    //añadir las siguientes dos líneas
    implementation platform('com.google.firebase:firebase-bom:28.4.1')
    implementation 'com.google.firebase:firebase-analytics'
    ...
}

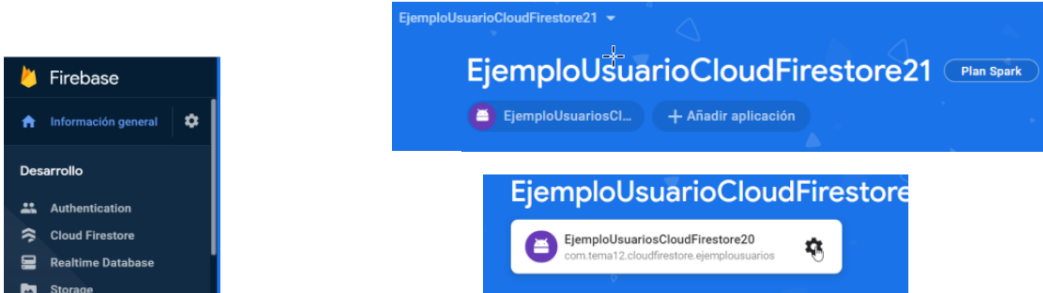
```

La dependència de firebase bom permetrà evitar afegir la versió a la resta de dependències de Firebase que incloguem en el nostre projecte, facilitant molt el treball.

Molt important no oblidar afegir en el SDK, el servei de Google Play

<input type="checkbox"/> Google Play Licensing Library	1	Not installed
<input checked="" type="checkbox"/> Google Play services	49	Installed
<input checked="" type="checkbox"/> Google USB Driver	12	Not installed
<input type="checkbox"/> Google Web Driver	2	Not installed

En la manera consola podem accedir a tots els nostres projectes, i podem comprovar l'aplicació o aplicacions afegides sobre un projecte (podem afegir més d'una).



Accedint a la configuració de l'aplicació o del projecte, podem veure informació sobre les claus, ANEU d'aplicació i altra. També podem descarregar de nou el .json de configuració dels serveis.

Totes les aplicacions per defecte es troben en manera desenvolupa i sota el pla gratis que suporta fins a 100 connexions concurrents, 1GB d'emmagatzematge i 10GB de transferència en el backend. Després de creada l'app ens dirigim a veure els seus detalls, podem entrar en diverses pestanyes que ens permetran treballar amb l'APP, però a nosaltres ens interessa la BD. Firebase ofereix dues solucions de bases de

dades en el núvol i accessibles per als clients que admeten sincronització en temps real:

- **Firestore DataBase** a vegades també nomenada Cloud Firestore, és la base de dades més recent de Firebase. Aprofita el millor de Realtime Database amb un model de dades nou, permet consultes més riques i ràpides, i l'escalament s'ajusta a un nivell més alt que Realtime Database.
- **Realtime Database** és la base de dades original de Firebase. És una solució eficient i de baixa latència destinada a les apps per a dispositius mòbils que necessiten estats sincronitzats entre els clients en temps real. És similar a l'anterior en el seu ús, per la qual cosa no l'explicarem.

Realtime Database	Cloud Firestore
Almacena datos como un gran árbol JSON	Almacena datos como colecciones de documentos
<ul style="list-style-type: none"><li>• Los datos simples son muy fáciles de almacenar.</li><li>• Los datos complejos y jerárquicos son más difíciles de organizar a gran escala.</li></ul>	<ul style="list-style-type: none"><li>• Los datos simples son fáciles de almacenar en documentos, que son muy similares a JSON.</li><li>• Los datos complejos y jerárquicos son más fáciles de organizar a escala, con subcolecciones dentro de los documentos.</li><li>• Necesita menos desnormalización y compactación de datos.</li></ul>

## Firestore DataBase

Base de dades NoSQL flexible, escalable i en el núvol a fi d'emmagatzemar i sincronitzar dades per a la programació en el costat del client i del servidor. Igual que [Firebase Realtime Database](#) manté les dades sincronitzades entre apps client a través d'agents d'escolta en temps real. El model de dades de [Firebase Firestore DataBase](#) admet estructures de dades flexibles i jeràrquiques.

Emmagatzema les dades en documents organitzats en col·leccions, els documents poden contindre objectes niats complexos, a més de subcolecciones. Està optimitzada per a emmagatzemar grans col·leccions de documents xicotets. El model de dades està format per documents, cada document conté un conjunt de parells clau-valor. Tots els documents s'han d'emmagatzemar en col·leccions, els documents poden contindre subcolecciones i objectes niats, i tots dos poden incloure camps primitius com strings o tipus d'objectes complexos com a llistes. Les col·leccions i els documents es creen de manera implícita en **Firestore DataBase**. Només has d'assignar dades a un document dins d'una col·lecció. Si la col·lecció o el document

no existeixen, **Firestore DataBase** els crea. Cada document està identificat per una clau única, que pot generar-se automàticament o que es pot afegir alhora que el document:

items	U8jXrWZQeWCGBcy7C7fd	items	dH6paTTRgtj61iI5etUU
<a href="#">+ Añadir documento</a>	<a href="#">+ Iniciar colección</a>	<a href="#">+ Añadir documento</a>	<a href="#">+ Iniciar colección</a>
4ASan8jPPxLWfY62ytEO	<a href="#">+ Añadir campo</a>	4ASan8jPPxLWfY62ytEO	<a href="#">+ Añadir campo</a>
U8jXrWZQeWCGBcy7C7fd	msg: "El primer mensaje"	U8jXrWZQeWCGBcy7C7fd	key: null
dH6paTTRgtj61iI5etUU	user: "xusa"	dH6paTTRgtj61iI5etUU	msg: "Como va todo?"
			user: "xusa33"

Són molt similars a JSON, de fet, bàsicament són JSON. Existeixen algunes diferències, per exemple: els documents admeten **tipus de dades addicionals** i la seua grandària es limita a 1 MB, però en general, pots tractar els documents com a registres JSON lleugers. Els documents viuen en col·leccions, que simplement són contenidors de documents. Per exemple, podries tindre una col·lecció anomenada users amb els diferents usuaris de la teua app, en la qual hi haja un document que represente a cadascun:

users

alovelace

first : "Ada"

last : "Lovelace"

born : 1815

aturing

first : "Alan"

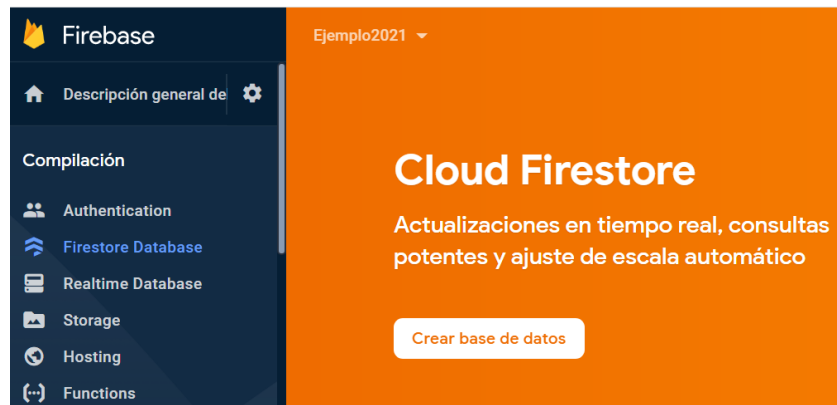
last : "Turing"

born : 1912



Usant l'exercici del qual hem creat el layout d'inici, afegirem el codi necessari perquè ens permeti interactuar amb Firebase. L'aplicació guardarà usuaris amb contrasenya en la base de dades amb **Firestore Cloud Firestore** i a més permetrà l'autenticació amb **Firestore Auth**.

En el projecte de Firebase que acabem de crear, afegirem una base de dades **Firestore DataBase**.



Haurem de seleccionar la ubicació de la nostra Base de dades, podem seleccionar ZURIC(europe-west6). Un element important i que encara no hem esmentat, són la Regles. La Firebase Cloud Firestore proporciona un llenguatge de regles flexibles basades en expressions i sintaxi similar a la de JavaScript <https://firebase.google.com/docs/firestore/security/get-started?authuser=0>, que permet definir fàcilment la manera en què les teues dades han d'estructurar-se i indexar-se, i el moment en què poden sotmetre's a lectura i escriptura. Aquestes regles les podrem configurar desde la pestanya RULES de la nostra BD,



Per defecte vindrien configurades per a manera producció, encara que per a iniciar-se podem seleccionar manera de prova (dura 30 dies).





```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if
        request.time < timestamp.date(2021, 1, 1);
    }
  }
}
```

Si ens haguérem confós o volguérem canviar les regles, caldria canviar el seu valor i sobretot **No oblidar Publicar**.

Abans de començar amb el codi d'accés a la BD, haurem d'afegir la dependència que ens permetrà fer-ho:

```
implementation 'com.google.firebase:firebase-firestore'
```

Per a referenciar a la nostra base de dades solament haurem de crear un objecte de tipus **Firestore**. L'arxiu que descarreguem en enllaçar l'app amb el projecte, és el que s'encarrega de tot el treball intern per a la connexió:

```
val firestore = FirebaseFirestore.getInstance()
```

Per a fer referència a una col·lecció existent o afegir-la en cas de no existir, utilitzarem el mètode **collection** amb el nom de la col·lecció com a argument.

```
firestore.collection("Usuarios")
```

Aquest mètode retorna una referència a la col·lecció seleccionada, i amb ell podrem afegir nous elements a ella (la col·lecció es crearà en afegir el primer document, en cas d'haver sigut creada anteriorment farà la referència solament).

Per a afegir un objecte (document) a la col·lecció ho podrem fer de dues maneres:

1. Deixant que la plataforma genere una clau aleatòria, per a això utilitzarem el mètode **add**.

```
firestore.collection("Usuarios").add(Usuario("Pepe", "correo@gmail.com"))
```

YFphu0kLMna9K3mvqU7u

ZqVp1P2p67KxXRnRsZdM

manuel@gmail.com

+ Añadir campo

correo: "correo@gmail.com"

usuario: "Pepe"

2. Afegint nosaltres la clau, aquesta ha de ser única perquè l'usuari s'afija.

```
firebaseFirestore.collection("Usuarios").document(usuario.correo).set(usuario)
```

maria@gmail.com



correo: "maria@gmail.com"

usuario: "Maria"

Per a inserir el valor del nou fill hem utilitzat un objecte de la classe Pojo Usuari, que ens haurem creat amb anterioritat, Firestore converteix automàticament els atributs amb els seus valors per a poder guardar-los correctament. **Cura haurem de tindre els atributs públics o usar getter i setter.**


```
class Usuario : Serializable {  
    lateinit var usuario: String  
    lateinit var correo: String  
  
    constructor() {}  
    constructor(usuario: String, correo: String) {  
        this.usuario = usuario  
        this.correo = correo  
    }  
}
```

El codi complet per a crear l'aplicació que ens afegirà un usuari cada vegada que premem el botó afegir, de manera que l'aneu de l'usuari siga el correu, serà el següent:

```

1  class MainActivity : AppCompatActivity() {
2      lateinit var firebaseFirestore: FirebaseFirestore
3      var listenerRegistration: ListenerRegistration? = null
4      override fun onCreate(savedInstanceState: Bundle?) {
5          super.onCreate(savedInstanceState)
6          setContentView(R.layout.activity_main)
7          firebaseFirestore = FirebaseFirestore.getInstance()
8          val usuarioET = findViewById<TextInputLayout>(R.id.usuario)
9          val correoET = findViewById<TextInputLayout>(R.id.correo)
10         val añadir = findViewById<Button>(R.id.anyadir)
11
12         val salida = findViewById<TextView>(R.id.salida)
13         añadir.setOnClickListener{
14             val usuario = Usuario(usuarioET.editText?.text.toString(),
15                                     correoET.editText?.text.toString())
16             firebaseFirestore.collection("Usuarios")
17                                     .document(usuario.correo)
18                                     .set(usuario)
19                                     .addOnSuccessListener {
20                     Toast.makeText(
21                         this,
22                         "Usuario Añadido",
23                         Toast.LENGTH_SHORT
24                     ).show()}
25                                     .addOnFailureListener { e ->
26                     Toast.makeText(
27                         this,
28                         "Error" + e.message,
29                         Toast.LENGTH_SHORT
30                     ).show()}
31             }
32         }
33     }

```

 **Nota:** Com es pot veuen en el codi del OnClick, afegim l'usuari recollit dels EditText creant com a clau el correu, com s'ha explicat anteriorment en el tema. A més als mètodes d'afegir objecte (set o add) se li poden assignaran diferents escuchadores per a comprovar l'estat del procés. **Línia 19 i 25.**

Per a [buscar un document en Cloud Firestore](#), existeixen diferents possibilitats basades en la clausula **Where**. Per exemple, si quisiéssim comprovar si l'aneu de l'usuari no està repetit i només en aqueix cas afegir-lo, podríem modificar l'anterior codi de la següent manera:

```

fun compruebaSiExisteYAnade(usuario: Usuario) {
    firebaseFirestore.collection("Usuarios")
        .whereEqualTo(FieldPath.documentId(), usuario.correo).get()
        .addOnCompleteListener(OnCompleteListener<QuerySnapshot?> {
            { task ->
                if (task.isSuccessful) {
                    if (task.result?.size() == 0) anyadeUsuario(usuario)
                    else Toast.makeText(
                        this,
                        "El correo ya existe, introduce uno nuevo",
                        Toast.LENGTH_LONG
                    ).show()
                } else {
                    Toast.makeText(this, task.exception.toString(),
                        Toast.LENGTH_LONG)
                        .show()
                }
            })
        })
}

fun anyadeUsuario(usuario: Usuario) {
    firebaseFirestore.collection("Usuarios")
        .document(usuario.correo).set(usuario)
        .addOnSuccessListener {
            Toast.makeText(
                this,
                "Usuario Añadido",
                Toast.LENGTH_SHORT
            ).show()
        }.addOnFailureListener { e ->
            Toast.makeText(
                this,
                "Error" + e.message,
                Toast.LENGTH_SHORT
            ).show()
        }
}

```

Si quisiéramos donar funcionalitat al botó **Eliminar**, podem fer alguna cosa semblança al següent. En aquest cas s'està eliminant per nom d'usuari, així que en el cas d'existir més d'un document amb el mateix nom, s'eliminaran tots.

```

fun Elimina(usuario: Usuario) {
    firebaseFirestore.collection("Usuarios")
        .whereEqualTo("usuario", usuario.usuario)
        .get()
        .addOnCompleteListener { task ->
            for (documento in task.result!!) documento.reference.delete()
        }
}

```


Si el que volem és controlar els canvis que ocorren en la BD, siga a través d'una aplicació o directament des de la consola de Firebase, haurem de registrar un listener del tipus `ListenerRegistration`, que s'inicialitzarà sobre la consulta que desitgem amb `addSnapshotListener`. En el següent exemple posem a escoltar tots els documents de la col·lecció `Usuarios`, mostrant en el `TextView` (que està sota els botons) el resultat de qualsevol modificació en qualsevol document de la col·lecció.

```
fun listarUsuarios() {
    val query = firebaseFirestore.collection("Usuarios")
    listenerRegistration = query.addSnapshotListener {value, error->
        if (error == null) {

            for (dc in value!!.documentChanges) {
                when (dc.type) {
                    DocumentChange.Type.ADDED -> salida.text =
                        "${salida.text}\nSe ha añadido:" +
                        "${dc.document.data}\n".trimIndent()
                    DocumentChange.Type.MODIFIED -> salida.text =
                        "${salida.text}\n Se ha modificado:" +
                        "${dc.document.data}\n".trimIndent()
                    DocumentChange.Type.REMOVED -> salida.text =
                        "${salida.text}\nSe ha eliminado:" +
                        "${dc.document.data}\n".trimIndent()
                }
            }
        }
        else Toast.makeText(this, "No se puede listar"+error,
            Toast.LENGTH_SHORT).show()
    }
}
```

Es pot realitzar la consulta sobre qualsevol element de la col·lecció després d'haver sigut seleccionat, de la següent manera:

```
query.whereEqualTo("correo", "manuel@gamil.com").addSnapshotListener{...}
```

 **Avís:** Un altre tema important a tindre en compte, és que la subscripció a una referència d'una base de dades de Firebase, és a dir, el fet d'assignar un listener a una ubicació de l'arbre per a estar al corrent dels seus canvis **no és una cosa gratuïta** des del punt de vista de consum de recursos. Per tant, és recomanable eliminar aqueixa subscripció quan ja no la necessitem. Per a fer això n'hi ha prou amb cridar al mètode `remove()` del listener registrat, quan no desitgem continuar escoltant.

```
override fun onDestroy() {  
    super.onDestroy()  
    listenerRegistration!!.remove()  
}
```

## FirestoreUI y RecyclerView

**FirestoreUI** ens permet associar fàcilment a un control **ListView** o **RecyclerView**, una referència a una llista d'elements d'una base de dades Firestore. D'aquesta manera, el control s'actualitzarà automàticament cada vegada que es produïska qualsevol canvi en la llista, sense haver de gestionar manualment per part nostra els esdeveniments de la llista, ni haver d'emmagatzemar en una estructura paral·lela la informació, ni haver de construir gran part dels adaptadors necessaris... en resum, estalviant moltes línies de codi i evitant molts possibles errors. Per a utilitzar **FirestoreUI** el primer que haurem de fer serà afegir la referència a la llibreria en el fitxer build.gradle del nostre mòdul principal. Cal tindre en compte que cada versió de **FirestoreUI** és compatible únicament amb una versió concreta de \*Firestore, per la qual cosa hem d'assegurar que les versions utilitzades de totes dues llibreries són coherents. En la pàgina de **FirestoreUI** teniu disponible la taula de compatibilitats entre versions. En el moment d'actualització d'aquestes anotacions:

```
implementation 'com.firebaseui:firebase-ui-firestore:8.0.0'
```

Aquesta llibreria ens proveeix d'un Adaptador derivat de **RecyclerView.ViewHolder** que gestionarà automàticament la càrrega de les dades que li passem com a referència en la vista assignada pel **Holder**, per a això el primer que haurem de crear és el ViewHolder personalitzat que gestione les dades de les nostres aplicació (reprenem l'exemple i implementem el botó llistar, podem crear un fragment o activity per a mostrar).

```

1  class Adaptador(options: FirestoreRecyclerOptions<Usuario>) :
2      FirestoreRecyclerAdapter<Usuario, Adaptador.Holder>(options),
3      View.OnClickListener {
4      private var listener: View.OnClickListener? = null
5      override fun onBindViewHolder(holder: Holder, position: Int,
6                                  model: Usuario) {
7          holder.bind(model)
8      }
9      override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
10          Holder {
11          val view: View = LayoutInflater.from(parent.context)
12              .inflate(R.layout.linea_recycler, parent, false)
13          view.setOnClickListener(this)
14          return Holder(view)
15      }
16      fun onClickListener(listener: View.OnClickListener?) {
17          this.listener = listener
18      }
19      override fun onClick(v: View?) {
20          listener?.onClick(v)
21      }
22      inner class Holder(v: View) : RecyclerView.ViewHolder(v) {
23          private val usuario: TextView
24          private val correo: TextView
25          fun bind(item: Usuario) {
26              usuario.text = item.usuario
27              correo.text = item.correo
28          }
29          init {
30              usuario = v.findViewById(R.id.usuario)
31              correo = v.findViewById(R.id.correo)
32          }
33      }
34  }

```

📌 **Nota:** Definim un adaptador com ja hem fet anteriorment, amb el Holder per a controlar les vistes de les línies del recycler **Línia 22 a 28**, on sobreescrivíem, obligatòriament, els dos mètodes i el constructor. He implementaríem els listener necessaris per al funcionament dels esdeveniments. La principal diferència és que l'Adaptador hereta de la classe **FirestoreRecyclerAdapter**. FirebaseUI ens facilita el llistat dels elements extrets de la BD proporcionant aquesta llibreria. A aquesta classe li hem d'indicar que els elements usats són: el ViewHolder personalitzat que acabem de crear i la classe java que utilitzem per a encapsular la informació de cada element de la llista

**FirestoreRecyclerAdapter<Usuari, Adaptador.Holder> .**

En crear un objecte d'aquesta classe, hem de passar-li un objecte de la mateixa llibreria, de tipus **FirestoreRecyclerOptions** .

```
val firestoreRecyclerOptions = FirestoreRecyclerOptions.Builder<Usuario>()
    .setQuery(query, Usuario::class.java).build()
```

Al qual hem de passar-li la següent informació:

- L'objecte classe del nostre Item (Usuari.class)
- La referència al node de la base de dades que conté la llista d'elements que volem mostrar en el control.

Ara només quedaria crear un objecte del tipus adaptador que ens hem creat, passant-li l'element **firebaseRecyclerOption** i ja tindríem quasi tot fet.

```
private fun cargarRecycler(query: Query) {
    val firestoreRecyclerOptions = FirestoreRecyclerOptions.
        Builder<Usuario>()
        .setQuery(query, Usuario::class.java).build()
    recyclerView = vista.findViewById(R.id.recycler)
    adapter = Adaptador(firestoreRecyclerOptions)
    //Click para eliminar elementos
    adapter!!.onClickListener{
        Toast.makeText(
            getActivity(),
            "Elemento eliminado" + recyclerView!!.
                getChildAdapterPosition(it),
            Toast.LENGTH_SHORT
        ).show()
        adapter!!.snapshots.getSnapshot(recyclerView!!.
            getChildAdapterPosition(it)).
            reference.delete()
    }
    recyclerView!!.adapter = adapter
    recyclerView!!.layoutManager = LinearLayoutManager(getActivity())
}
```

No haurem d'oblidar iniciar el escoltador de l'adaptador en començar l'aplicació i tancar-lo en acabar.

```
override fun onStart() {
    super.onStart()
    adapter!!.startListening()
}
```

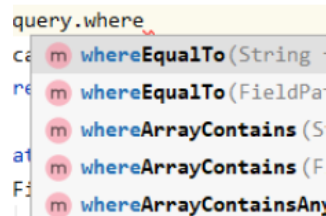
```
override fun onStop() {
    super.onStop()
    adapter!!.stopListening()
}
```



## Filtrat i ordenació

El primer que hem de tindre en compte és que en les BD de Firebase no tindrem totes les facilitats d'ordenació i filtrat que solen trobar-se en bases de dades SQL tradicionals. Per a les consultes podem trobar la informació que necessitem en el següent enllaçe <https://firebase.google.com/docs/firestore/query-data/queries>.

Per a realitzar consultes s'utilitza qualsevol dels mètodes **where** que indiquen que tipus de consulta necessitem:



Usant la col·lecció de l'exemple que ens trobem en l'enllaç de la pàgina de documentació de firestore, amb l'expressió següent recuperariem tots els documents que tenen una població menor a 100000 habitants.

```
citiesRef.whereLessThan("population", 100000);
```

Es poden enllaçar cerques amb **where**, de la següent manera:

```
citiesRef.whereEqualTo("state", "CA").whereLessThan("population", 100000);
```

En aquest cas es buscaran tots els documents d'estat igual a CA i del resultat d'aqueixa cerca s'extrauran només aquells que tinguen una població menor a 100000.

També està l'opció de buscar dins d'una col·lecció directament. Si alguns dels nostres documents tenen un membre que és una col·lecció, podem realitzar la consulta sobre aquest element usant alguna de les sobrecàrregues de **whereArrayContains**

```
citiesRef.whereArrayContains("regions", "west_coast");
```

En les dades 'regions' es refereix a un array amb els noms de les regions a les quals pertany cada ciutat, per la qual cosa amb la consulta anterior s'extrauran totes les ciutats que en aqueix array tinguen l'entrada 'west\_coast', si la tenen repetida només apareixeran una vegada en els resultats.

Si el que volem fer és ordenar les dades obtingudes, tenim tota la informació en la url <https://firebase.google.com/docs/firestore/query-data/orderlimitdata?hl=es> S'utilitzarà qualsevol de les sobrecàrregues del mètode **orderBy**

```
query.or
: m orderBy(String field)
: m orderBy(FieldPath fieldPath)
: m orderBy(String field, Direction directi
it m orderBy(FieldPath fieldPath, Direction
```

Per exemple, es poden combinar més d'un **orderBy** per a realitzar la consulta que necessitem. En aquest cas s'ordena pel nom de l'estat i a partir d'ací per número de població i en ordre descendent.

```
citiesRef.orderBy("state").orderBy("population", Direction.DESENDING);
```

Una clàusula **orderBy** també filtra a la recerca de l'existència del camp donat. El conjunt de resultats no inclourà documents que no continguin el camp corresponent. Com és d'esperar, també es poden ordenar les dades després de realitzar una consulta **where**, però amb la condició que ha de ser sobre el mateix camp pel qual s'ha fet la cerca.

```
citiesRef.whereGreaterThan("population", 100000).orderBy("population");
```

Si amb **orderBy** es pot especificar l'ordre de classificació de les dades, amb **limit** pots limitar la quantitat de documents recuperats.

*Retorna els noms de les tres primeres ciutats*

```
citiesRef.orderBy("name").limit(3);
```

*Retorna els noms de les tres últimes ciutats*

```
citiesRef.orderBy("name", Direction.DESENDING).limit(3);
```

Altres cursors de consultes que estan disponibles són:

- **startAt** -> La consulta només retornarà els elements el valor dels quals siga igual o superior a la dada passada com a paràmetre.

*Ciutats amb nombre d'habitants 100000 i ordenats per número de població*

```
db.collection("cities")
  .orderBy("population")
  .startAt(100000);
```

- **startAfter** -> Idèntic a l'anterior però per a valors superiors, no inclou els documents amb valor igual.

- **endAt** -> La consulta només retornarà els elements el valor dels quals siga igual o inferior a la dada passada com a paràmetre.

```
db.collection("cities")
  .orderBy("population")
  .endAt(100000);
```

*Ciutats amb nombre d'habitants <= 100000 i ordenats per número de població*

- **endBefore** -> Idèntic a l'anterior però per a valors inferiors, no inclou els documents amb valor igual.

També podem utilitzar diversos criteris de filtrat en la mateixa consulta, és a dir podem combinar diversos dels mètodes anteriors per a obtenir només el rang d'elements necessari.

```
Query next = db.collection("cities")
    .orderBy("population")
    .startAfter(lastVisible)
    .limit(25);
```

## Firestore Auth. Autenticació

[Firestore Authentication](#) proporciona serveis de backend, SDK fàcils d'usar i biblioteques d'UI ja elaborades per a autenticar als usuaris en l'app. *Ens iniciarem en el seu ús modificant el projecte anterior, per a això crearem un altre Fragment que serà el primer que s'iniciï en l'aplicació. Ens permetrà crear un compte o autenticar-nos amb una ja existent, si l'autenticació és correcta es carregarà el fragment amb el qual s'iniciava el nostre projecte anteriorment.*

Si volem autenticar-nos com a usuaris, tenim diverses opcions: **email/password, google, telèfon, Facebook, twitter, github, anonymous**. Nosaltres veurem l'exemple d'email/password i el de anonymous, en la resta d'autenticacions podeu seguir la documentació de FireBase.

El primer pas seria afegir la dependència d'autenticació, que com tenim inclòs la de \*firebase-\*bom, no necessitarà número de versió:

```
implementation 'com.google.firebase:firebase-auth'
```

*Tendremos que codificar un layout que permeti introduir el usuario y la contraseña para poder logearnos.*

Usuario

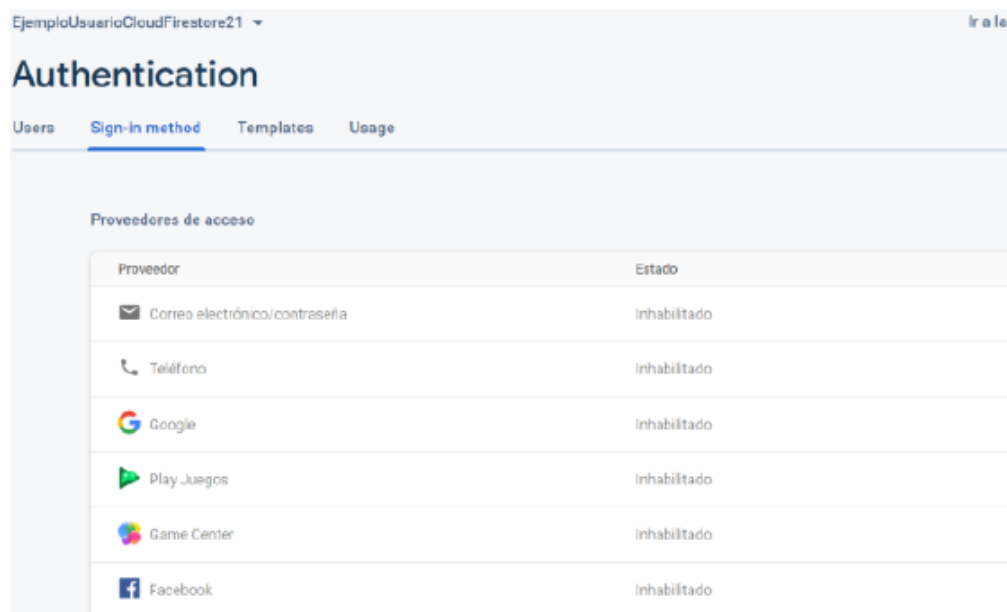
Contraseña

SIGNIN

SIGNUP

ANONIMO

El següent serà activar els proveïdors amb els quals volem iniciar sessió. Ens anem a l'opció \*Authentication i dins d'aquest a la pestanya Mètodes d'inici de sessió. Haurem de seleccionar tots dos casos i habilitar l'estat.



Una vegada afegit els proveïdors, tindrem dues maneres de crear els usuaris: **a través de la consola i des de l'aplicació**. Des de la consola ens haurem d'anar a la pestanya usuaris, ací podrem crear els usuaris que necessitem. Els usuaris anònims apareixeran solament amb l'aneu únic que assigna Firebase a cada usuari creat.

L'altre cas serà en codificar l'aplicació. Per a tots els casos d'autenticació necessitem una instància de `FirebaseAuth.getInstance()`, a partir d'aquesta podrem afegir els escuchadores necessaris:

El següent pas serà decidir si volem permetre crear usuaris nous, o solament logearnos amb algun existent. En la nostra aplicació farem les dues coses, i fins i tot

ens logearem com a anònims. El codi apareix a continuació, i un enllaç amb tota la informació <https://firebase.google.com/docs/auth/android/password-auth>

```
//////// Crear usuario nuevo y iniciar sesión
btCrear.setOnClickListener{
    FirebaseAuth.getInstance().createUserWithEmailAndPassword(
        user.editText!!.text.toString(), password.editText!!
        .text.toString()
    )
    .addOnCompleteListener(requireActivity(),
        OnCompleteListener<AuthResult?> { task ->
            if (task.isSuccessful) {
                Toast.makeText(getActivity(), "Usuario creado",
                    Toast.LENGTH_SHORT).show()
                iniciarFragmen(task.result?.getUser()?.
                    getEmail()?.split("@")!![0])
            } else Toast.makeText(
                getActivity(),
                "Problemas al crear usuario" +task.exception,
                Toast.LENGTH_SHORT
            ).show()
        })
    })
}
return view
```

```
////////Iniciar sesión con usuario y contraseña
btIniciar.setOnClickListener {
    FirebaseAuth.getInstance().signInWithEmailAndPassword(
        user.editText!!.text.toString(), password.editText!!
        .text.toString()
    )
    .addOnCompleteListener(requireActivity(),
        OnCompleteListener<AuthResult?> { task ->
            if (!task.isSuccessful) {
                Toast.makeText(
                    getActivity(),
                    "Authentication failed:" + task.exception,
                    Toast.LENGTH_SHORT
                ).show()
            } else iniciarFragmen(task.result?.getUser()?.
                getEmail()?.split("@")!![0])
        })
    })
}
```

```

////////// Iniciar sesión con anónimo
btAnonimus.setOnClickListener{
    FirebaseAuth.getInstance().signInAnonymously().addOnCompleteLis
    (
        requireActivity(),
        OnCompleteListener<AuthResult?> { task ->
            if (!task.isSuccessful) {
                Toast.makeText(
                    getActivity(),
                    "Authentication failed:" + task.exception,
                    Toast.LENGTH_SHORT
                ).show()
            } else iniciarFragmen("anonymous")
        }
    )
}

```

Com podem veure en el codi, el que es fa és usar qualsevol dels mètodes que hàgem triat (login, anònim o afegir usuari) sobre la instància de **FirebaseAuth**. Amb aquests passos tindrem l'autenticació controlada, i podrem seguir amb la nostra aplicació.

 **Resol l'exercici dels exemples fins a aconseguir un correcte funcionament**