

---

# Computação na nuvem

## ISEL – LEIRT / LEIC / LEIM

### Java API, de acesso ao **Google *Firestore***

Exemplos de criação de documentos

José Simão [jsimao@cc.isel.ipl.pt](mailto:jsimao@cc.isel.ipl.pt) ; [jose.simao@isel.pt](mailto:jose.simao@isel.pt)

Luís Assunção [lass@isel.ipl.pt](mailto:lass@isel.ipl.pt) ; [luis.assuncao@isel.pt](mailto:luis.assuncao@isel.pt)

- Documentação (java docs) das Java API dos serviços GCP
  - <https://googleapis.dev/java/google-cloud-clients/latest/index.html>
- Java API, de acesso ao **Firestore**
  - <https://googleapis.dev/java/google-cloud-firestore/latest/index.html>
- Dependência Maven

```
<dependency>  
  <groupId>com.google.cloud</groupId>  
  <artifactId>google-cloud-firestore</artifactId>  
  <version>3.1.0</version>  
</dependency>
```

# Criar Conta de Serviço

- Ao criar a conta de serviço para acesso ao *Firestore* usa-se uma permissão (role) "*Cloud Datastore Owner*"

Create service account

✓ Service account details — 2 Grant this service account access to project

## Service account permissions (optional)

Grant this service account access to CN1920-PJ01 so that it has permission to complete specific actions on the resources in your project. [Learn more](#)

Role  
Cloud Datastore Owner ▼

Condition

[Add condition](#)



Full access to Cloud Datastore.

+ ADD ANOTHER ROLE

CONTINUE

CANCEL

A BD *Firestore* pode ser usada em modo *Datastore* (versão anterior) ou modo nativo (versão usada nos slides)

# Iniciar o acesso ao serviço *Firestore*

```
InputStream serviceAccount = new FileInputStream(KEY_JSON);
```

```
GoogleCredentials credentials =  
    GoogleCredentials.fromStream(serviceAccount);
```

*pathname* do ficheiro *.json*  
com chave privada da conta  
de serviço

```
// Ou Caso se use a variável GOOGLE_APPLICATION_CREDENTIALS
```

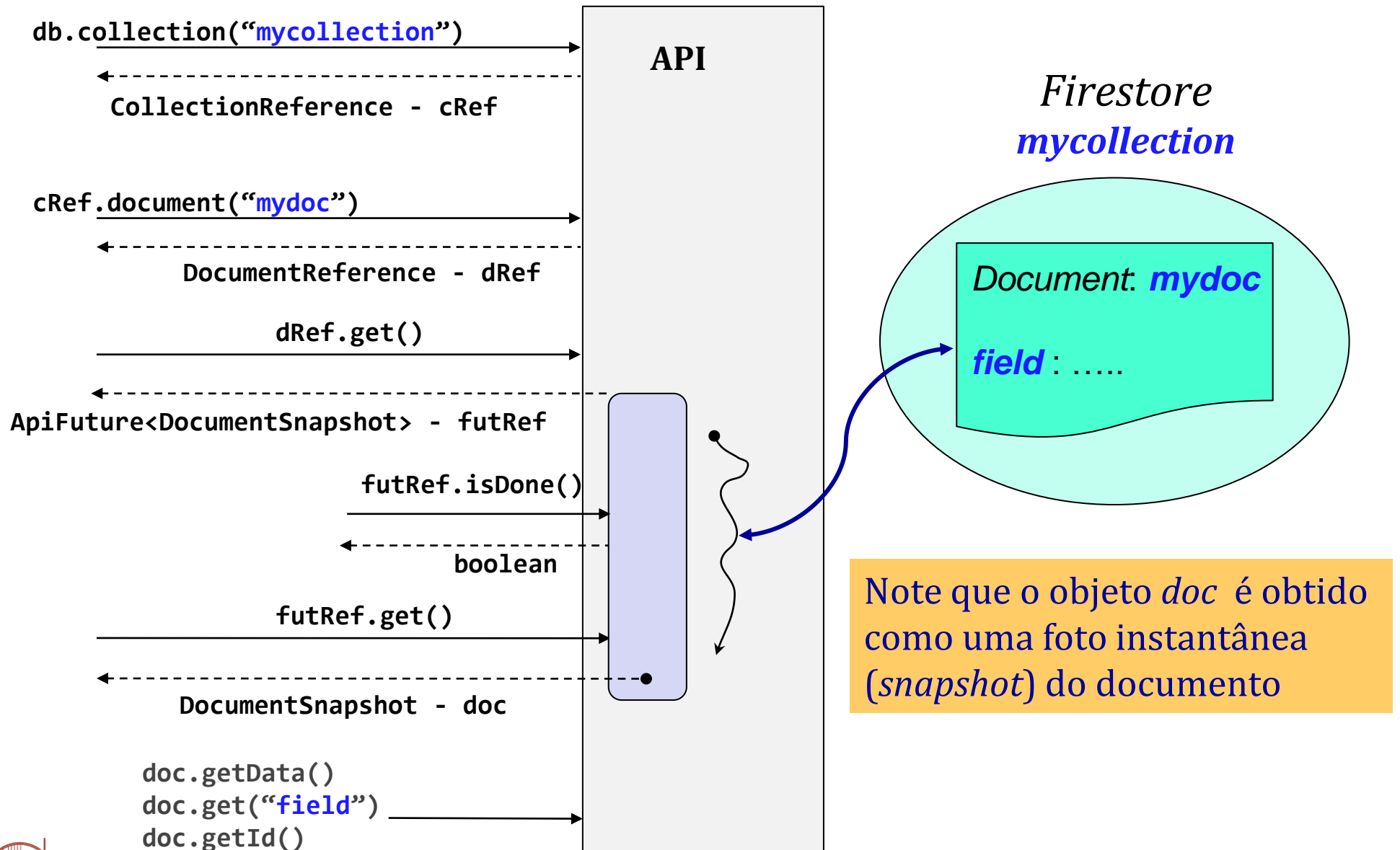
```
GoogleCredentials credentials =  
    GoogleCredentials.getApplicationDefault();
```

```
FirestoreOptions options = FirestoreOptions  
    .newBuilder().setCredentials(credentials).build();
```

```
Firestore db = options.getService();
```

- Note que 1 projeto GCP só pode ter 1 base de dados *Firestore*

# Modelo de chamadas à API



# Inserir/atualizar com *map*

---

```
CollectionReference colRef = db.collection("Users");
DocumentReference docRef = colRef.document("Bill-Gates");
HashMap<String, Object> map = new HashMap<String, Object>();
map.put("first", "Bill");
map.put("last", "Grates");
map.put("born", 1955);
```

```
ApiFuture<WriteResult> result = docRef.create(map); //Creates new document
result.get();
```

*// ou*

```
ApiFuture<WriteResult> result = docRef.set(map); //Overwrites a document
result.get();
```

*// update field*

```
map.put("last", "Gates");
result = docRef.update(map);
result.get();
```

# Inserir documentos a partir de objetos

```
public class OcupacaoTemporaria {  
    public int ID;  
    public Localizacao location;  
    public Evento event;  
}
```

```
public class Evento {  
    public int evtID;  
    public String nome;  
    public String tipo;  
    public Date dtInicio;  
    public Date dtFinal;  
    public Licenciamento licenciamento;  
    public Map<String, String> details;  
}
```

```
import com.google.cloud.firestore.GeoPoint;  
  
public class Localizacao {  
    public GeoPoint point;  
    public Coordenadas coord;  
    public String freguesia;  
    public String local;  
}
```

```
public class Coordenadas {  
    public Double X;  
    public Double Y;  
}
```

```
public class Licenciamento {  
    public String code;  
    public Date dtLicenc;  
}
```

# Inserir documento a partir de objeto

```
CollectionReference colRef = db.collection("ocupa-espacos");
OcupacaoTemporaria ocup = new OcupacaoTemporaria();
ocup.ID = 1111;
ocup.location = new Localizacao();
ocup.location.point = new GeoPoint(-9.143645, 38.753404);
ocup.location.freguesia = "Alvalade";
ocup.location.local = "Praça de Alvalade";

ocup.event = new Evento(); ocup.event.evtID = 1017;
ocup.event.nome = "Programa do Indie Junior";
ocup.event.tipo = "Publicitário";
```

Campos do documento ficam com os nomes e valores de cada campo do objeto 'ocup'.

Alguns tipos têm tratamento especial, ex: *GeoPoint* e *Date*.

O nome do documento (*Document ID*) é único e é atribuído pela aplicação.

```
DocumentReference docRef = colRef.document("DocID+ocup.ID);
```

```
//asynchronously overwrite data using Futures
```

```
ApiFuture<WriteResult> result = docRef.set(ocup);
```

```
System.out.println("Update time:" + result.get().getUpdateTime());
```

```
// criar o documento com Document ID gerado automaticamente pelo Firestore
DocumentReference docRef=colRef.document();
ApiFuture<WriteResult> futres = docRef.set(ocup);
```



# Listagem de documentos de uma coleção

---

```
CollectionReference cref = db.collection("ocupa-espacos");  
Iterable<DocumentReference> allDocs = cref.listDocuments();  
for (DocumentReference docref : allDocs) {  
    ApiFuture<DocumentSnapshot> docfut = docref.get();  
    DocumentSnapshot doc = docfut.get();  
    // Time at which this document was last updated  
    Timestamp updateTime = doc.getUpdateTime();  
    System.out.println(updateTime + ":doc: " + doc.getData());  
}
```

# Ler campo ou objeto a partir de um documento

---

```
String ID="1111";  
DocumentReference docRef = db.collection("ocupa-espacos").document(ID);  
ApiFuture<DocumentSnapshot> future = docRef.get();  
DocumentSnapshot document = future.get();
```

*//ler campo do documento*

```
GeoPoint coord = document.getGeoPoint("location.point");  
System.out.println(coord.toString());
```

*//ler objeto: obtém campos do documento para campos  
// com o mesmo nome na classe*

```
OcupacaoTemporaria ocup = document.toObject(OcupacaoTemporaria.class);  
System.out.println(ocup.location.point.toString());
```

# Apagar campos e documentos

---

```
DocumentReference docRef = db.document("ocupa-espacos/1111");
```

```
// apagar campo
```

```
Map<String, Object> updates = new HashMap<>();  
updates.put("location.coord", FieldValue.delete());  
ApiFuture<WriteResult> writeResult = docRef.update(updates);  
System.out.println("Update time : " + writeResult.get());
```

```
// apagar documento
```

```
ApiFuture<WriteResult> resFuture = docRef.delete();  
WriteResult res = resFuture.get();
```

# Query simples

*// Single query*

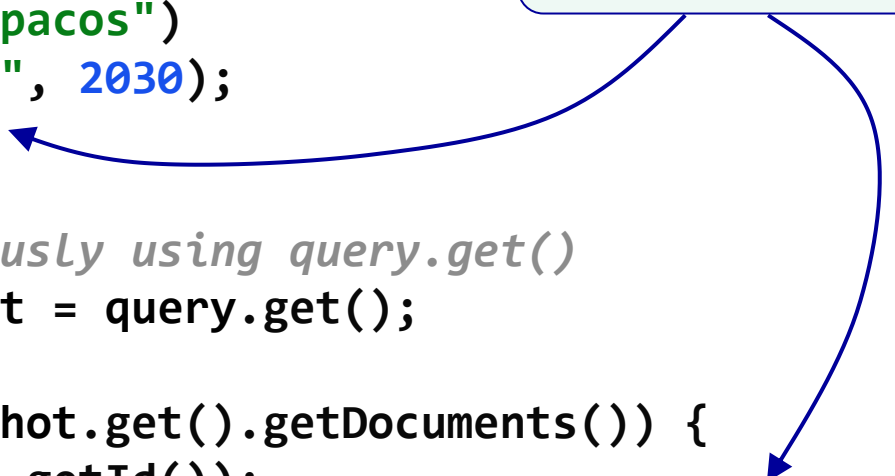
```
Query query = db.collection("ocupa-espacos")  
                .whereGreaterThan("ID", 2030);
```

*// retrieve query results asynchronously using query.get()*

```
ApiFuture<QuerySnapshot> querySnapshot = query.get();
```

```
for (DocumentSnapshot doc: querySnapshot.get().getDocuments()) {  
    System.out.print("Doc id: " + doc.getId());  
    System.out.println(" Freguesia: " + doc.get("location.freguesia"));  
}
```

Nomes de  
campo (*field*)



# Interrogações simples de campos complexos

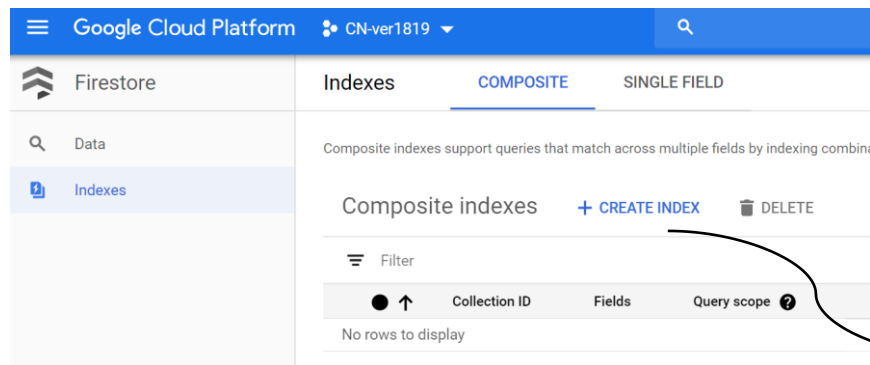
```
FieldPath fp = FieldPath.of("location", "freguesia");
Query query = db.collection("ocupa-espacos").whereEqualTo(fp, "Alvalade");
ApiFuture<QuerySnapshot> querySnapshot = query.get();
for (DocumentSnapshot doc: querySnapshot.get().getDocuments()) {
    System.out.print("Doc id: " + doc.getId() + " @ " + doc.get("location"));
}
```

caminhos para campos

```
fp = FieldPath.of("location","coord");
HashMap<String,Double> cor = new HashMap<String, Double>() {
    { put("X",-9.143645364765742); put("Y",38.75340432875235); }
};
query = db.collection("ocupa-espacos").whereEqualTo(fp, cor);
querySnapshot = query.get();
for (DocumentSnapshot doc: querySnapshot.get().getDocuments()) {
    System.out.print("Doc id: " + doc.getId() + " @ " + doc.get("location"));
}
```

# Interrogações compostas

- As *interrogações simples* são suportadas por índices criados automaticamente
- As *interrogações compostas* podem necessitar de um **índice composto**, construído pelo programador



### Create a composite index

Tip: Instead of manually creating indexes, you can simply run queries in your code and look out for error messages with links to create the required indexes. [Learn more](#)

Collection ID \*  
ocupespacos

### Fields to index

Field paths	Index options	
location.freguesia	Ascending	^ v
ID	Ascending	^ v

+ ADD FIELD

Creation time can vary based on the amount of data being indexed

SAVE INDEX CANCEL

# Interrogações compostas com índice composto

*// Composed query*

```
FieldPath fpath = FieldPath.of("location", "freguesia");

Query query = db.collection("ocupa-espacos")
    .whereEqualTo(fpath, "Misericórdia")
    .whereLessThan("ID", 2100);

ApiFuture<QuerySnapshot> querySnapshot = query.get();
for (DocumentSnapshot doc: querySnapshot.get().getDocuments()) {
    System.out.println(doc.getId()+":Doc:"+doc.getData());
}
```

❖ Se não existir um índice, ao executar este código gera uma exceção:

FAILED\_PRECONDITION: The query requires an index. You can create it here:

[https://console.firebase.google.com/v1/r/project/cn2122-jsla-geral/firestore/indexes?create\\_composite](https://console.firebase.google.com/v1/r/project/cn2122-jsla-geral/firestore/indexes?create_composite) . . .

É boa prática seguir o *link* e aceitar na consola GCP a criação do índice sugerido