

PART 1: INTRODUCTION TO FIRESTORE AND CREATING OUR FIRST DATABASE



Lecture Overview

2

- Firestore Database
 - ▣ Overview of Document Driven Databases
 - ▣ Creating our first database
- Connecting the database to our Firebase functions
 - ▣ Writing our comment data to the database
 - ▣ Reading our comment data from the database
- All will be tested using POSTMAN

Purpose of the lecture

3

- The goal is to introduce you to Firestore, from the point of view of using it as a backend for your applications. The majority of the discussion will be practically focussed, with little theory concerning more advanced database concepts such as sharding, normalisation, concurrency, BSON, locking writes/reads etc.
- It will be a basic introduction on how to get a database connected to your applications.

Architecture

4

Clients



Firestore

**Functions
(Node.js)**

Firestore (Database)

Call API endpoint

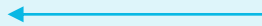


url:api

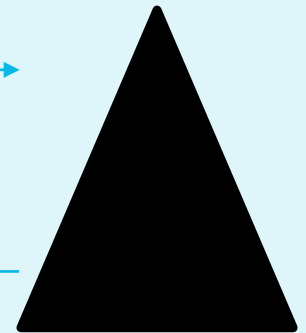
Query data



Return JSON



Return JSON



What is Firestore?



Cloud
Firestore

5

- ❑ Firestore is a Document Driven Database.
- ❑ Documents follow a **property:value** format
 - ❑ JSON
- ❑ Scalable, highly performant and document oriented.
- ❑ The databases tend to scale more easily horizontally.

Database concepts

6

- Records in Firestore are known as “**Documents**”
 - These *documents* are just JSON data
- Documents are grouped into “**Collections**” which are equivalent to tables in relational databases
- Queries are still queries, however there is NoSQL!

SQL to Firestore Terminology

7



Database



Database

Table



Collection

Record/Tuple/Row



Document

Column

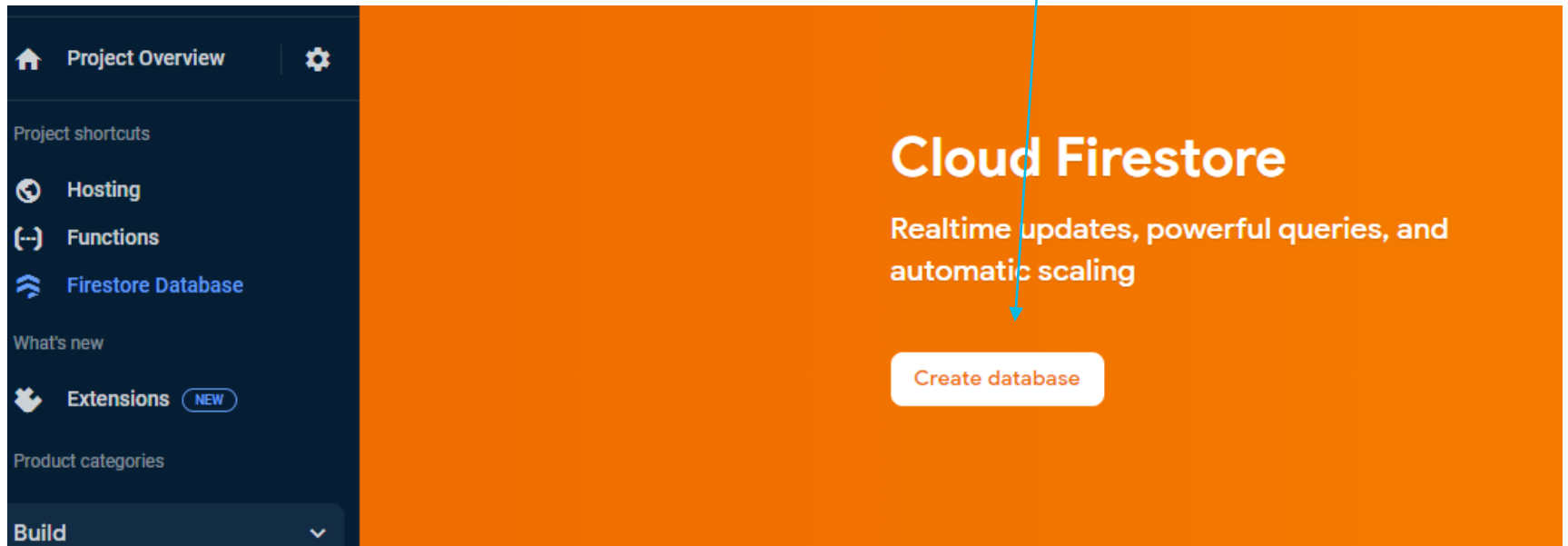


Field

Creating our first database

8

Login to the Firebase dashboard, click on Firestore and then “Create database”




Open in production mode

9

❑ Start in production mode

Create database ✕


1 Secure rules for Cloud Firestore — 2 Set Cloud Firestore location

After you define your data structure, you will need to write rules to secure your data.
[Learn more](#) 

Start in production mode
Your data is private by default. Client read/write access will only be granted as specified by your security rules.

Start in test mode
Your data is open by default to enable quick setup. However, you must update your security rules within 30 days to enable long-term client read/write access.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if false;
    }
  }
}
```

 All third party reads and writes will be denied

Enabling Cloud Firestore will prevent you from using Cloud Datastore with this project, notably from the associated App Engine app

Cancel **Next**

Choosing a region

10

- The latency should be fairly low so the default region will be fine, but if you want to place it in Europe please select it in the dropdown and then click enable

Create database ×

1 Secure rules for Cloud Firestore — 2 Set Cloud Firestore location

Your location setting is where your Cloud Firestore data will be stored.

⚠ After you set this location, you cannot change it later. Also, this location setting will be the location for your default Cloud Storage bucket. [Learn more](#)

Cloud Firestore location

eur3 (europe-west) ▼

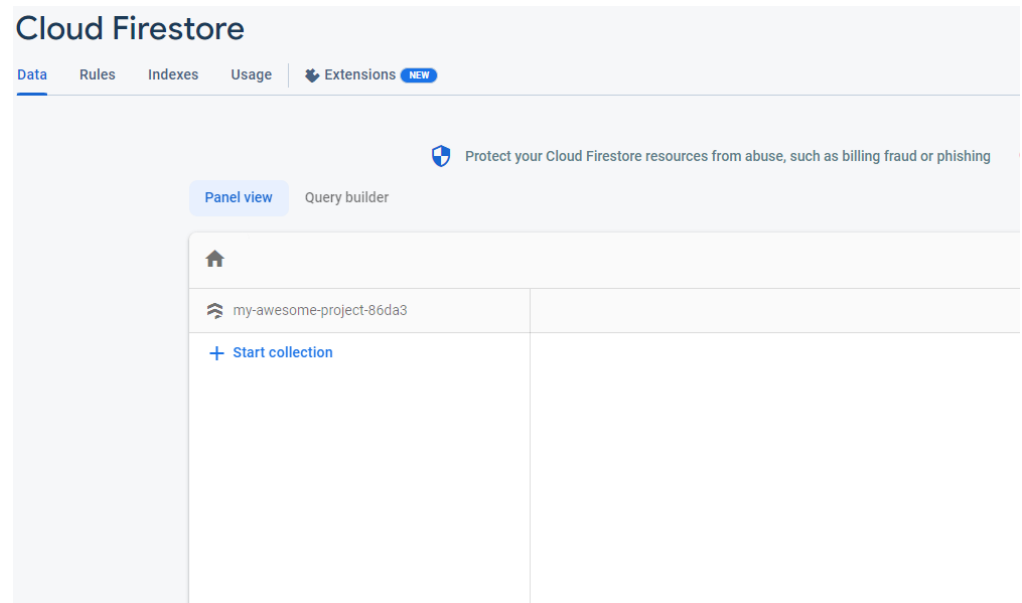
Enabling Cloud Firestore will prevent you from using Cloud Datastore with this project, notably from the associated App Engine app

[Cancel](#) [Enable](#)

Database is now created

11

- You can create a collection and add documents manually via this web interface. But the next step is to connect to it with our functions and read/write data.



Summary Overview

12

- ~~Firestore Database~~
 - ~~Overview of Document Driven Databases~~
 - ~~Creating our first database~~

- Connecting the database to our Firebase functions
 - Writing our comment data to the database
 - Reading our comment data from the database

Writing data to the database

13

- To motivate data writing we will reuse the postcomments function
- This is known as “Creating” a document
- I’ll create a new document every time the postcomments function is called and save it in the database
- <https://firebase.google.com/docs/firestore>

Firestore admin

14

- ❑ Firestore provides an admin library to allow your server code (functions) to run in an authenticated mode
- ❑ This means your code can connect to the database, create docs, delete docs, update etc. all securely

```
const functions = require('firebase-functions');  
const admin = require('firebase-admin');  
admin.initializeApp();
```

Promise – More async hell

15

- In ES6 a new concept was added to JavaScript to handle **Callback hell**

- These are called promises

- What's the difference between callbacks and promises?
 - ▣ Callback is passed as an argument
 - ▣ Promise is something that is achieved or completed in the future.
 - Promise is an object, **then()** method (if promise is fulfilled) and **catch** (if promise is rejected)

Code examples

16

```
asyncFunc(result => {  
  console.log(result);  
});
```

Callback



```
const promise = asyncFunc(()=>{  
  return new Promise...  
});  
promise.then(result => {  
  console.log(result);  
});
```

Promise



Adding a document

17

```
const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp();

exports.postcomments = functions.https.onRequest((request, response) => {
  // 1. Receive comment data in here from user POST request
  // 2. Connect to our Firestore database
  return admin.firestore().collection('comments').add(request.body).then(()=>{
    response.send("Saved in the database");
  });
});
```

Using POSTMAN POST to the fn

18

▶ Post Comments


POST ▼ https://us-central1-my-cool-web-app-37271.cloudfunctions.net/postcomments

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON ▼

```
1 {  
2   "@handle" : "EndaB",  
3   "comment" : "This is my second comment"  
4 }
```

Body Body Cookies Headers (9) Test Results

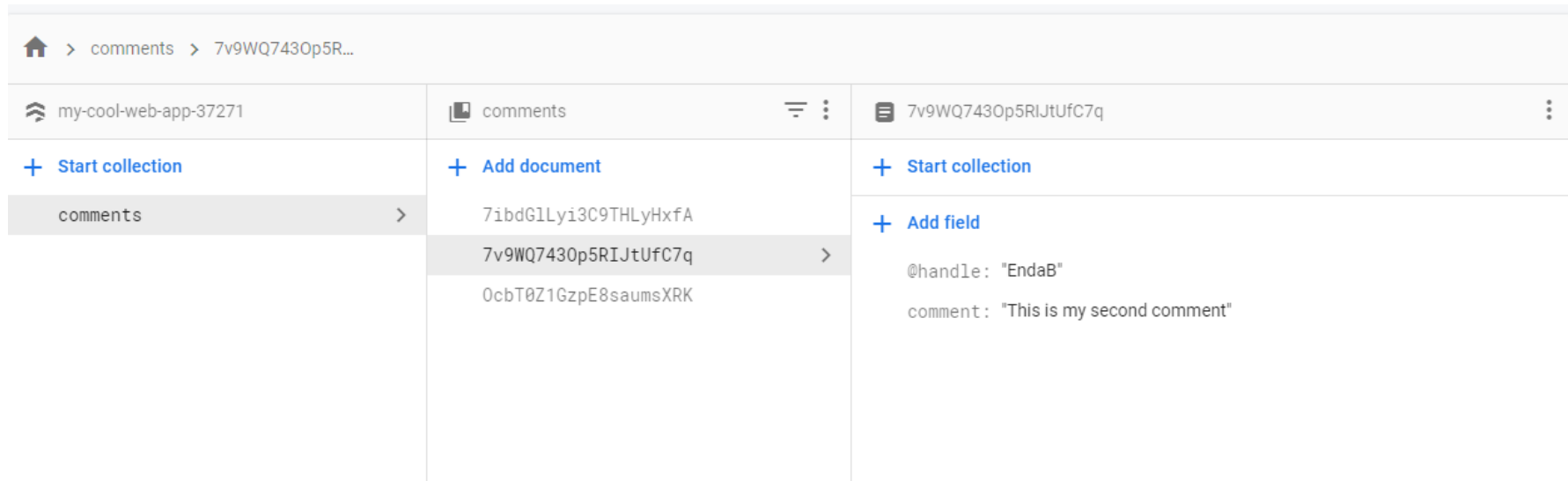
Pretty Raw Preview Visualize HTML ▼ 

1 Saved in the database

Check the database to see if it saved

19

- ❑ If you check on Firebase you should now see your comment



The screenshot shows the Firebase console interface. The breadcrumb path is `home > comments > 7v9WQ7430p5R...`. The left sidebar shows the project `my-cool-web-app-37271` and the `comments` collection. The main area displays the `comments` collection with three documents: `7ibdG1Ly13C9THLyHxfA`, `7v9WQ7430p5RIJtUfC7q` (selected), and `0cbT0Z1GzpE8saumsXRK`. The right pane shows the details for the selected document, including the `@handle` field with value `"EndaB"` and the `comment` field with value `"This is my second comment"`.

my-cool-web-app-37271	comments	7v9WQ7430p5RIJtUfC7q
+ Start collection	+ Add document	+ Start collection
comments >	7ibdG1Ly13C9THLyHxfA	+ Add field
	7v9WQ7430p5RIJtUfC7q >	@handle: "EndaB"
	0cbT0Z1GzpE8saumsXRK	comment: "This is my second comment"

Reading our documents

20

```
exports.getcomments = functions.https.onRequest((request, response) =>
{
  // 1. Connect to our Firestore database
  let myData = []
  admin.firestore().collection('comments').get().then((snapshot) => {

    if (snapshot.empty) {
      console.log('No matching documents. ');
      response.send('No data in database ');
      return;
    }

    snapshot.forEach(doc => {
      myData.push(doc.data());
    });

    // 2. Send data back to client
    response.send(myData);
  })
});
```

myCoolApp/Functions/index.js

Test the function with POSTMAN

21

Post Comments


GET ▼ <https://us-central1-my-cool-web-app-37271.cloudfunctions.net/getcomments>

Params Authorization Headers (7) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

This re

Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize JSON ▼ 

```
1  [
2    {
3      "@handle": "JohnD",
4      "comment": "This is my first comment"
5    },
6    {
7      "comment": "This is my second comment",
8      "@handle": "EndaB"
9    },
10   {
11     "comment": "This is my first comment",
12     "@handle": "EndaB"
13   }
14 ]
```

```

const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp();

exports.postcomments = functions.https.onRequest((request, response) => {

    // 1. Receive comment data in here from user POST request
    // 2. Connect to our Firestore database
    admin.firestore().collection('comments').add(request.body);
    response.send("Saved in the database");
});

exports.getcomments = functions.https.onRequest((request, response) => {

    // 1. Connect to our Firestore database
    let myData = []
    admin.firestore().collection('comments').get().then((snapshot) => {

        if (snapshot.empty) {
            console.log('No matching documents. ');
            response.send('No data in database');
            return;
        }

        snapshot.forEach(doc => {
            myData.push(doc.data());
        });

        // 2. Send data back to client
        response.send(myData);
    });
});

```

OrderBy

23

- So far when reading comments from the database we have not given any consideration to their order
- Perhaps it would be useful to order them by postdate or perhaps by the number of likes etc.
- To do this we need to modify our Firebase functions postcomments and getcomments to order the comments

Creating comments - postcomments

24

- The Firestore database supports a timestamp field, which we can use to store the date and time each comment was posted.
- Once this is recorded on each document we can return the comments to the user in order of their post date/time.

Posting comments

25

```
exports.postcomment = functions.https.onRequest((request, response) => {  
  console.log("Request body", request.body);  
  // Create a timestamp to add to the comment document  
  const currentTime = admin.firestore.Timestamp.now();  
  request.body.timestamp = currentTime;  
  
  admin.firestore().collection('comments').add(request.body).then(()=>{  
    response.send("Saved in the database");  
  });  
});
```

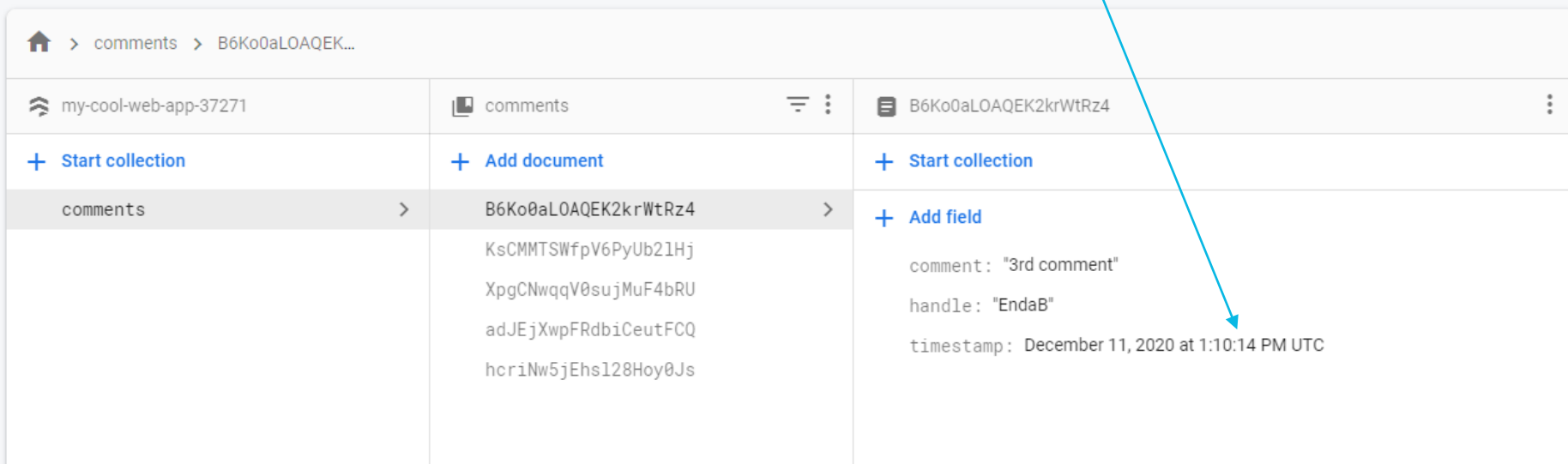
Don't forget to hit *firebase deploy*
once you have made your changes

myCoolApp/functions/index.js

Check database

26

- When you post a comment you should now see a timestamp beside each comment



The screenshot shows the Firebase console interface. The breadcrumb path is `home > comments > B6Ko0aLOAQEK...`. The left sidebar shows the project `my-cool-web-app-37271` and the `comments` collection. The main area displays a document with ID `B6Ko0aLOAQEK2krWtRz4`. The document content is:

```
{
  "comment": "3rd comment",
  "handle": "EndaB",
  "timestamp": "December 11, 2020 at 1:10:14 PM UTC"
}
```

A blue arrow points from the text in the list above to the `timestamp` field in the document.

Ordering documents by timestamp

27

- We now modify the get comments firebase function to order the comments by timestamp

```
exports.getcomments = functions.https.onRequest((request, response) => {  
  
  // 1. Connect to our Firestore database  
  let myData = []  
  admin.firestore().collection('comments').orderBy('timestamp').get().then((snapshot) => {  
  
    if (snapshot.empty) {  
      console.log('No matching documents.');      response.send('No data in database');      return;  
    }  
  
    snapshot.forEach(doc => {  
      myData.push(doc.data());  
    });  
  
    // 2. Send data back to client  
    response.send(myData);  
  
  });  
});
```

Lecture Overview

28

- Firestore Database
 - ▣ Overview of Document Driven Databases
 - ▣ Creating our first database

- Connecting the database to our Firebase functions
 - ▣ Writing our comment data to the database
 - ▣ Reading our comment data from the database