PART 1: INTRODUCTION TO FIRESTORE AND CREATING OUR FIRST DATABASE

Lecture Overview

- 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

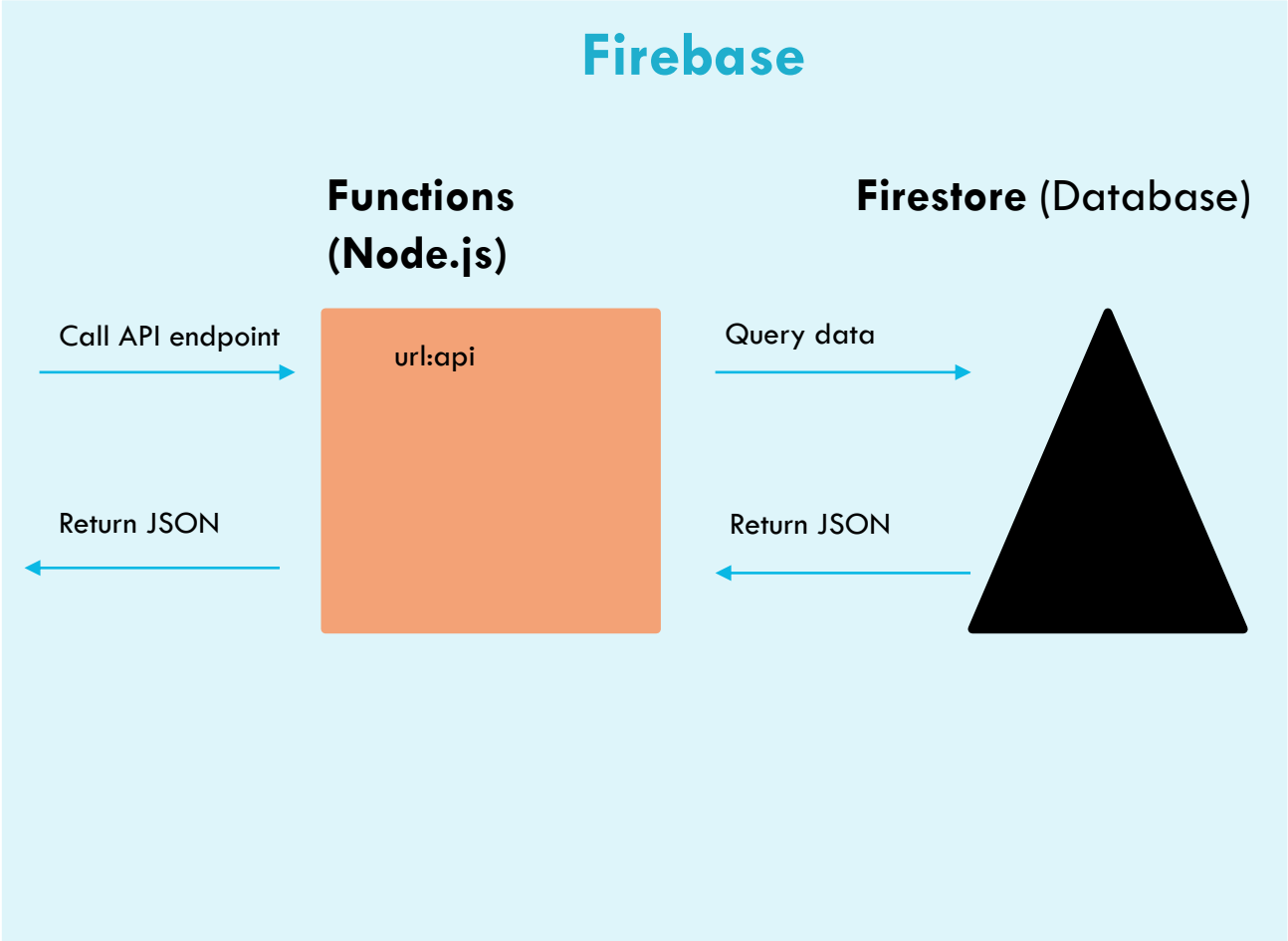
Purpose of the lecture

- 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

Clients



What is Firestore?

- 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

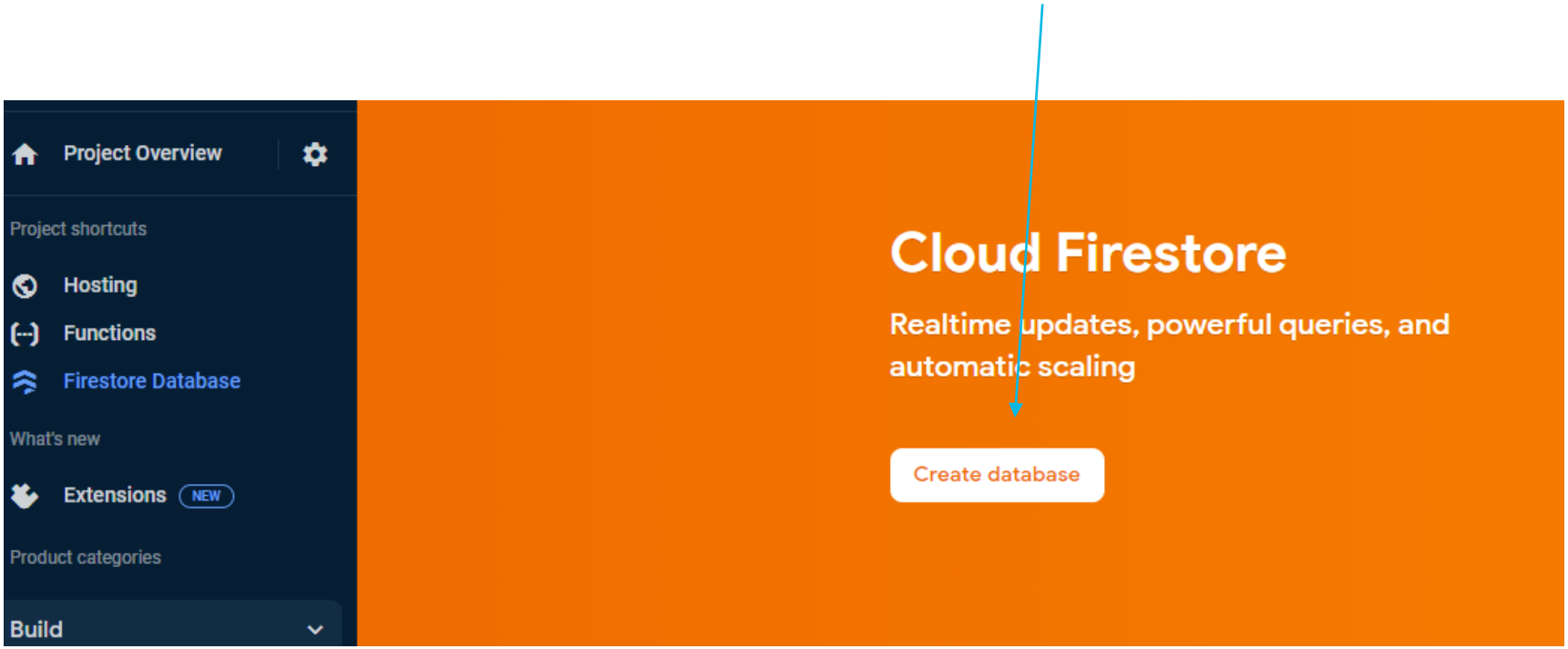
- Records in Firestore are known as "**Documents**"
  - These *documents* are just JSON data
- Documents are grouped into **"Collections"** which are equivalent to tables in relational databases
- Queries are still queries, however there is NoSQL!

SQL to Firestore Terminology

Database	⟷	Database
Table	⟷	Collection
Record/Tuple/Row	⟷	Document
Column	⟷	Field

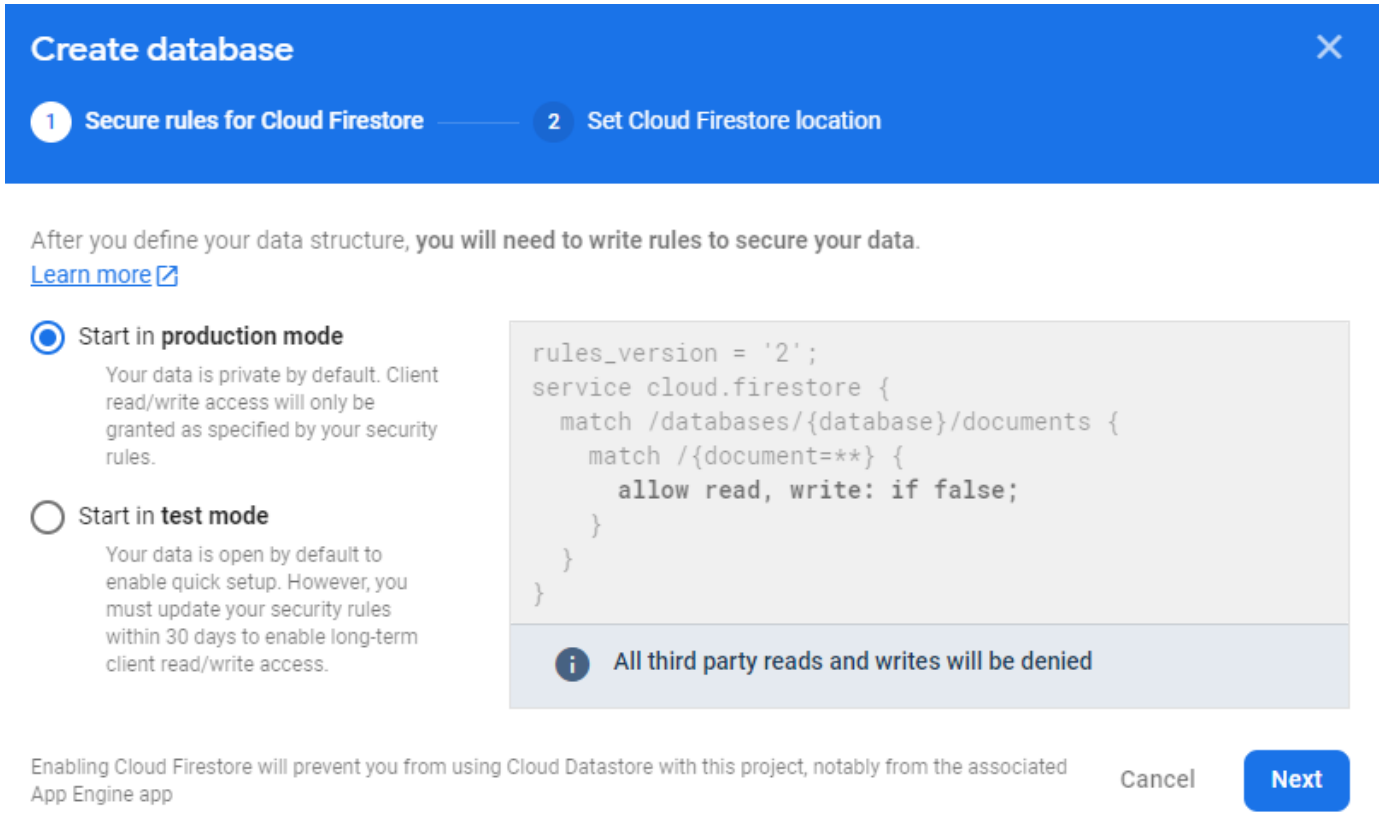
Creating our first database

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



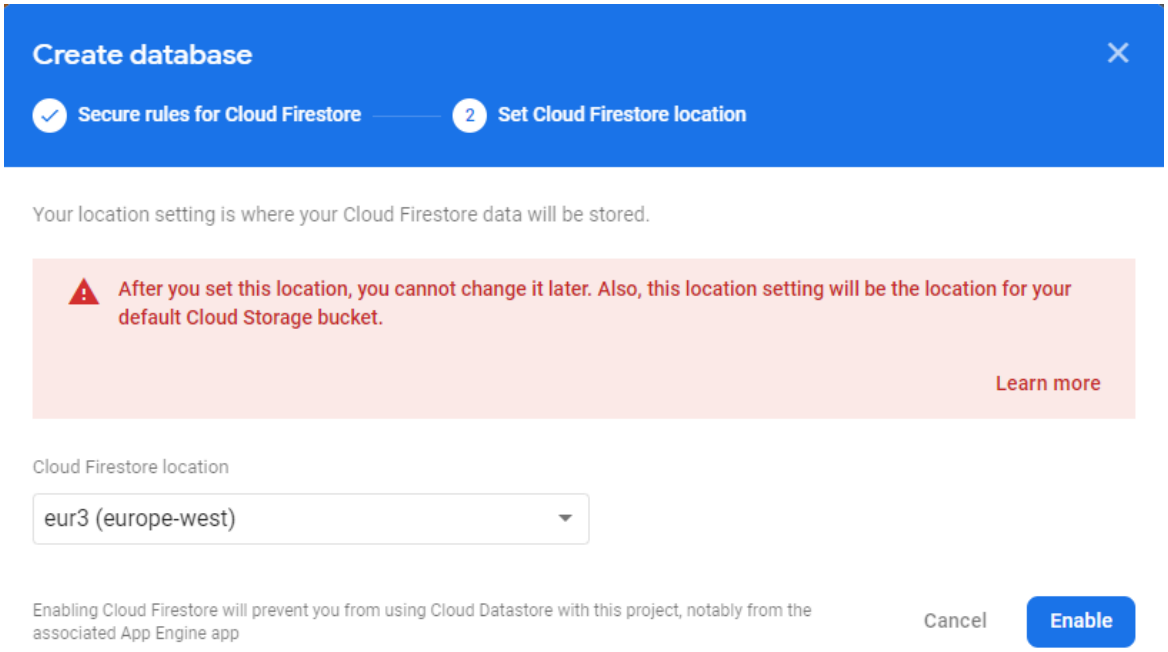
Open in production mode

- Start in production mode



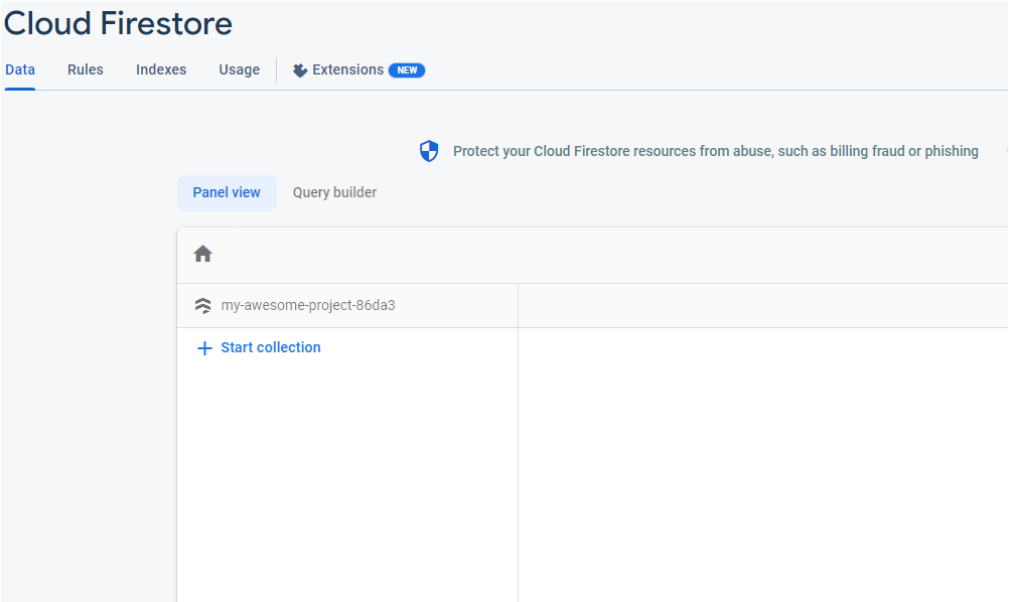
Choosing a region

- 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



Database is now created

- 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

- ~~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

- 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

Firebase admin

- Firebase 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

- 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


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

Callback

Promise

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

Adding a document


```
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");
  });
});
```

myCoolApp/Functions/index.js

Using POSTMAN POST to the fn

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 }
```

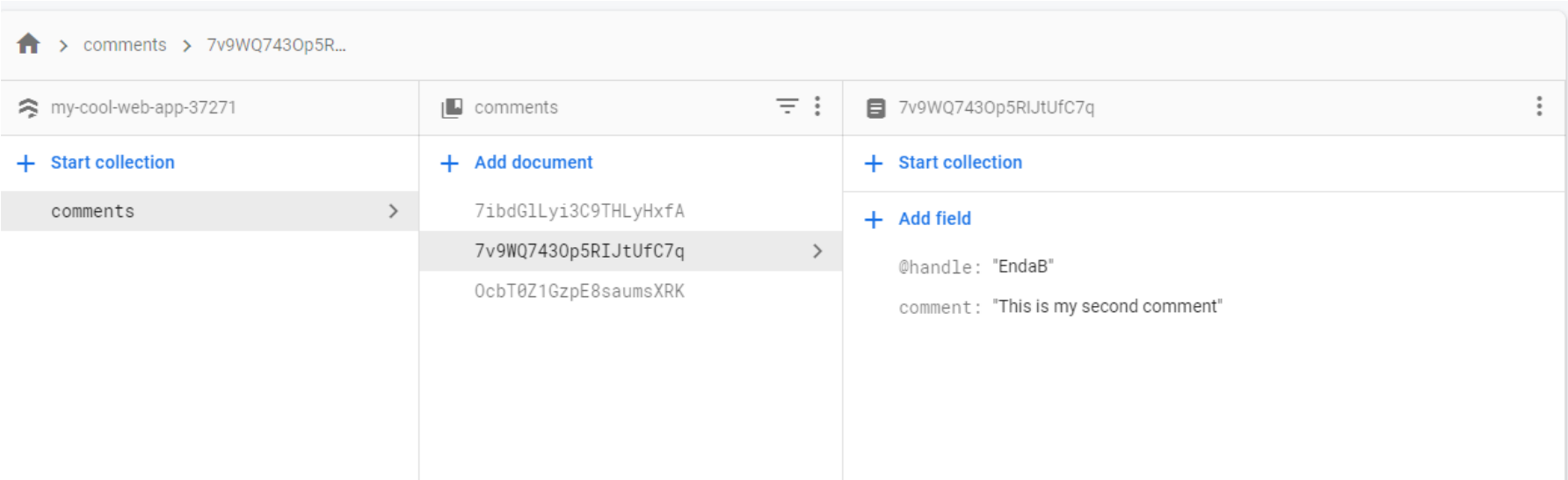
Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize HTML

```
1 Saved in the database
```

Check the database to see if it saved

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



Reading our documents


```
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

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


```
[
    {
        "@handle": "JohnD",
        "comment": "This is my first comment"
    },
    {
        "comment": "This is my second comment",
        "@handle": "EndaB"
    },
    {
        "comment": "This is my first comment",
        "@handle": "EndaB"
    }
]
```

```
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);
   })
});
```

myCoolApp/Functions/index.js

OrderBy

- 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

- 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

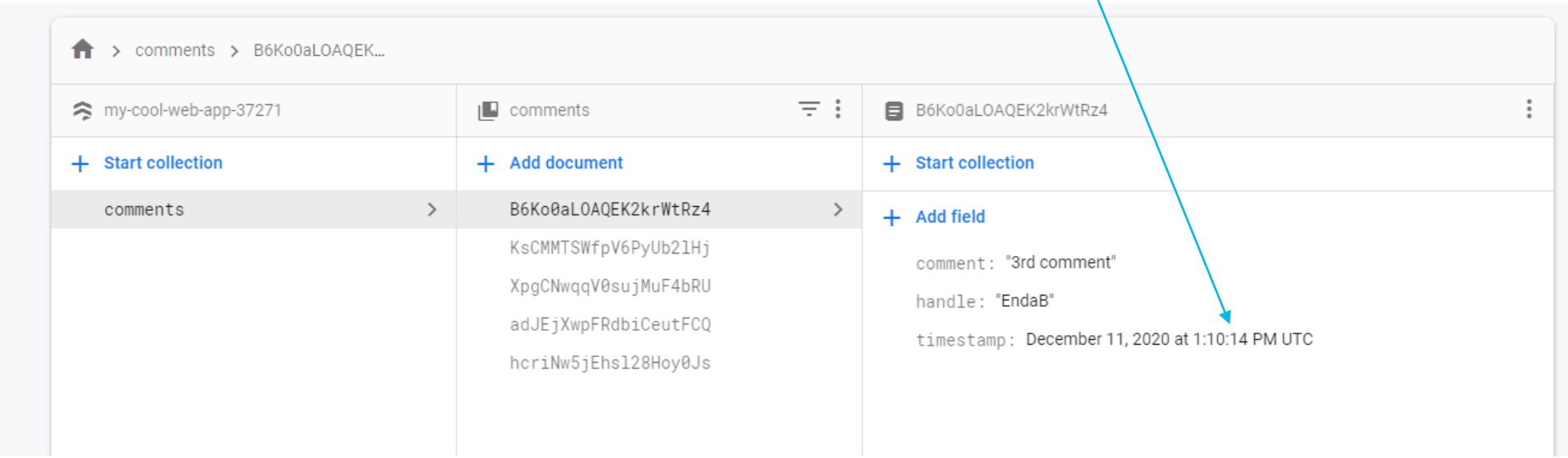

```
exports.postcomment = functions.https.onRequest((request, response) => {
        console.log("Request body", request.body);
                // Create a timestamp to add to the comment document
        const currentTime = admin.firestore.Timestamp.now();
        request.body.timestamp = currentTime;

        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

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



Ordering documents by timestamp

- 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);
   })
});
```

myCoolApp/functions/index.js

Lecture Overview

- 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