Serverless Full-Stack

This lesson discusses the differences between the popular Node and Express stack and a serverless Firebase stack. You will see that Firebase is not only capable of doing the same full-stack tasks but easier to implement.

WE'LL COVER THE FOLLOWING

When you implement Firebase as your back-end, your back-end is cloud-based. It's provided to you as a service and there is no need for you to personally manage the server.

Comparing NodeJS, Express, and MongoDB with Firebase

In order to fully illustrate the differences between these stacks, I will show you the code that serves up data from a traditional back-end. This can also be referred to as an API (application programming interface). Then I will show you how to make an asynchronous request to that back-end in order to fetch data for use on the front-end. In order to use this particular stack,i.e. MongoDB with Node.js and Express, you must have a front-end and a back-end.

After those examples, I show you how to do the exact same thing using Firebase's Cloud Firestore database. In that example, you will get the same results by implementing just the front-end. Technically, there is a back-end (server) but because you don't manage it directly, it is serverless.

How to get data from a

How to get data from a Node.js/Express/MongoDB back-end

First, make an API route using Express (line 7). You can make a query to the database from inside it (lines 8-15). Lastly, send the data from the query result back through the route as the response (line 14).

As of this writing, this was one of the most popular stacks around. Since Node.js was released in 2009 it was possible, for the first time, to write the back-end in the same language as the front end. That language being JavaScript.

Check out the code below which uses Node/Express/MongoDB back-end to get data:

```
var express = require('express');
var app = express();

// MongoDB connection details would go here

// Express GET route
app.get('/all-friends', () => {
  // Get the documents collection
  const collection = db.collection('friends');
  // Find some documents
  collection.find({}).toArray((err, docs) => {
    assert.equal(err, null);
    // Send the query response
    res.send(docs);
  });
});

app.listen(process.env.PORT || 3000);
```

backend code example using NodeJS, Express and MongoDB

Let's see what happens on the front-end. Make an asynchronous request to the back-end route in order to get the data on the front-end.

```
// window fetch method is making an asynchronous request to that endpoint
fetch('/all-friends')
// data in the first .then() is the whole response with headers and all
.then(data => data.json())
// data in this .then() is cleaned up and in usable JSON format
.then(data => console.log(data));
```

back-end code example

How to Get Data From Firebase Cloud Firestore

Be aware that the example code you see below is very short. It may look weird at first but you will quickly get used to it and appreciate how easy it is to implement as you learn more about Firebase.

Using only a few lines of code, we can make a real-time connection to our database!

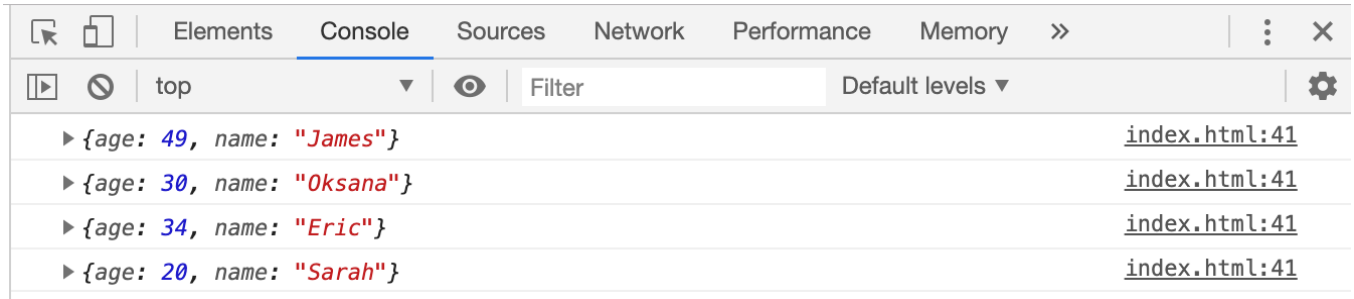
```
// listen to the collection of friends
db.collection("friends")
.onSnapshot(function(snap) {
// comes back in a way that is iterable with a forEach() loop
        snap.forEach(function(doc) {
                // doc.data() is each document coming from the friends collection
                console.log(doc.data());
        });
});
```

firebase code example

There is no step two! We defined the endpoint to query from our front-end while also making an asynchronous request. This also keeps the line of communication open, so if your database changes, all connected clients will be updated without needing to make another request.

A look at the Firebase data

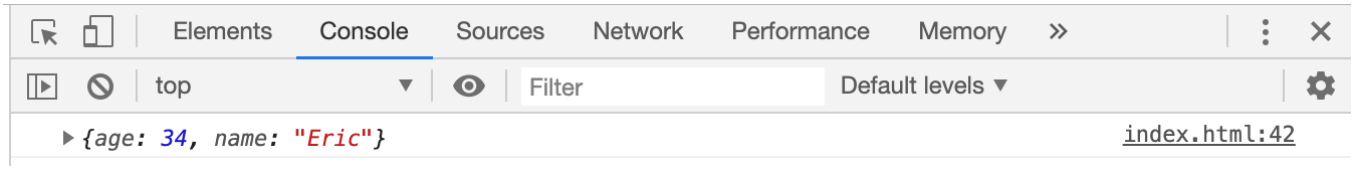
Since this course isn't about Node, Express, or MongoDB, we will not discuss that stack anymore. Let's start getting to know Firebase better by looking at the response that was logged in the console from our code example above.



Using the `where` Clause

To specify what we want back from Cloud Firestore, we make use of the "where" clause.

```
    db.collection("friends")
// By adding the "where" clause we have control over what data comes back
    .where('name', '==', 'Eric')
.onSnapshot(function(snap) {
    snap.forEach(function(doc) {
        console.log(doc.data());
    });
});
```



What we just covered is a short example of how to work with Cloud Firestore. It's not the traditional way of doing things but it's short, effective, and fast to implement. Keep in mind that updating, deleting and other CRUD like queries are just as easy. I will teach you a lot more about Cloud Firestore in the upcoming lessons.

In the next lesson, I will show you full demos of the apps you will build throughout the course.