Now in the last lesson, we've updated our chat screen so that we're able to start typing messages over

here, so we could say 'what's up?'

And when I hit send, that message goes straight into our database on our Firebase Cloud

Firestore. And we can see our new message pop up like so right here.

So that part is done and we're able to send messages over to our database.

But how can we retrieve data from our database and use it to populate our app?

So this is what we're going to do in this lesson. First of all just underneath where we have our get

CurrentUser,

I'm going to create a new method called getMessages and this method is going to go into our Firestore.

So I'm going to tap into that Firestore I've got and tap into the right collection, which is the one

called messages. And then I'm going to call the getDocuments method and you can see that this returns

a future query snapshot.

Now this query snapshot is a data type that comes from Firebase and it's a snapshot of the data as we

have it in our current collection.

So if I go ahead and call this method and I want to use the result of this method, then of course I have

to await for it.

So let's save the data that we get back

as a final and we'll call it our messages.

And of course I have to mark this method with async if I want to use that await and we'll put a equal

sign in between those two parts

so we don't get any warnings or errors there.

Now if we tap into these messages and we go into the documents, you can see that we can get a list

of documents snapshots.

So those documents refer to what we have in here, see how we can add a document.

Well we've already got three of them

and those belong to the messages collection.

So if we want to grab all of these documents then we have to tap in to the messages.documents.

And because this is a list, in order to be able to view individual items in the list, then we need to

use a FOR loop.

So let's create a for in loop,

one of my favorite types of FOR loops, and I'm going to create a new variable. We'll call it our var message

in messages.ocuments.

So this of course is the list that we're looping through

and each of those items in there should be a message.

And once we've gotten a hold of the message, then we can go ahead and print that document snapshot.

data.

So now this should give us that key value pair, the string which refers to the field and the value which

refers to the text.

Now we just have to be able to trigger this method and I'm going to do that right here where we currently

are doing the sign out because this is the part that's easiest the comment out.

So we're going to get rid of the sign out bits of the code and you can change this icon if it makes

it easier for you.

I'm just going to leave it as it is because I'm only using it to demo this method right now.

So I'm going to call that method that I created just now called getMessages.

And now if we hit save to hot reload our app, then we should be able to click on this button which I've

linked to getMessages and once I've clicked on that, I should be able to see it inside my console.

Now mind you, the final version of the app isn't going to have people clicking on the logout button to

view their chat messages.

I'm just temporarily adding the code here so we can easily test it out.

So what's printed here is all of the messages that are currently in my messages collection. The way that

the code is working at the moment is it gets a hold of all the documents inside my messages collection,

it creates a single message out of all of the message documents and we print out each one of those and

the data that's associated with it.

So this is how we're getting hold of all our documents printed in here.

Now that's all very well and good but if I go in to say here and I add a new document will leave the

document ID blank and will keep the fields the same as what we had before,

so sender and text. And I'll just put in some sort of sender value and a text value in here and hit save

and you can see that it's gone into my collection.

Now imagine this was not me posting a message manually through the Firebase dashboard but instead it

was a chat message sent by another user.

If another user posts a chat message to our database, it would show up just like this.

But if we take a look over here, in order to get the updated value I have to call that method again.

I have to click on the button again before I can retrieve all of my messages.

Effectively I'm actively pulling data out of my Firebase database.

Yes, pulling our database and retrieving the chat messages does work.

But can you think of some downsides of using get documents to build our chat app?

Why is this not the way that we would want to build it?

Well imagine that you're chatting with your friend.

In that case you want to see your friends messages show up right away

right?

And that means you want your instant messenger to deliver those messages,

well instantly,

right?

So to achieve this, the app would probably have to call get documents two times a second or even more.

Now imagine you say good night to your friend and you'll continue chatting in the morning.

At this point the last thing you want is your chat app checking in with Firebase two times a second for

new messages.

That's why we need a different approach to having our app pull Firebase for a new chat messages. What

if we could get the data to be pushed to our app when new data gets added

from your conversation partner or somebody else who has the app installed? You kind of want the message

to be automatically pushed over and shown on your screen right?

That's kind of how WhatsApp and how most messaging apps work.

So how can we do this?

Well in order to do this, we need to know about something called streams.

Now I'm going to comment out this method getMessages and I'm going to create a different method.

So down here I'm going to create a method called messagesStream and this method is going to again not

take any inputs, but it is also going to be an asynchronous method because I'm going to use it to listen

for the stream of messages that's coming over from Firebase instead of waiting on firestore.collection

messages .getDocuments which is the method that is going to give us back a future.

So it's a future query snapshot that we get back.

Well instead the method I'm going to use in here is again firestore.collections to find the right

collection,

the messages one. And then I'm gonna use a different method and it's a method called snapshots.

And this method instead of a future query snapshot, it returns a stream of query snapshots.

And this is almost kind of like a list of futures, it's a whole bunch of futures.

And by subscribing to that stream, I'm going to listen to all of the changes that happen to this particular

collection.

And as the documentation tells us, when there's a new document that goes into my messages collection,

I am now subscribed and I will be notified of any new results.

So how do we handle these streams?

Well the easiest way to think about it is to think of it almost as a list. And as you saw previously

we needed to use a FOR loop to loop through all the messages in the list of message documents.

We have to loop through our snapshots which is basically kind of like a list of future objects.

So at any time point, a new message could be created and it could be added to our messages collection

and we are now subscribing ourselves to that stream.

So the syntax that I'm going to write is of course again await.

So I'm waiting for this to complete before I can use it.

And I'm also going to add the FOR loop here.

So it's await for and inside this for in loop, I'm going to save the result of this stream to something

that I'll call a snapshot.

And that for loop is going to have the same syntax,

so it's for var snapshot in, this bunch of snapshots.

And inside this curly brace of the FOR loop, I will now get access to the snapshot. And the snapshot has

something called .documents.

So same as what we've got before,

it's a list of document snapshots. so I can again use this code that I've got from before to loop through

my snapshots although I might have to change some the words so it'll be snapshot.document.

And each of these will be a message. And instead of calling getMessages, I'm going to call my new method

which is the messages stream and we hit save and we open up our console.

And now when I go and hit the cross sign and call that method messagesStream, then we should start subscribing

to the messages in that collection. And we should be printing out each message into the console.

So let's hit that button, run that message and just as before we're getting all of the messages that

currently live in our messages collection.

But watch this.

If I go ahead and add a document say on Firebase and it's going to again use the same field sender and

text, so sender will again just be angela@gmail.com and value will be 'Coming over the air from

firebase console.'

Here we're simulating another user sending a chat message.

All right.

Now I'm going to hit save which will save this document into my collection and I'm doing this through

the Firebase dashboard but it's all going to the same database that I'm now subscribed to in my app.

So now if I go back to the console, you'll see that it triggered a rerun of this code where because I've

subscribed myself to listen for changes to my collection of messages, as soon as there was a change it's

coming out over here and we're getting all of the results printed

again. So now instead of pulling data from our Firebase database, we're effectively pushing data over.

So this is really cool right?

Our app is now listening for changes to the database.

It's not a case of pulling Firebase and checking for messages as we did in the first part of lesson.

Now Firebase is notifying our app of any new messages via the stream of data snapshots.

This is exactly the kind of behavior that you would want from a chat application.

So in order to understand what's going on here we have to understand a bit more about how streams work.

And that's exactly what we're going to do in the next lesson.

So head over there for a deep dive on, 'what exactly are streams and how do they differ from futures?'

