Now in the last lesson we talked about Dart streams and how they allow us to listen for data that might　be updated at any moment in time, so that we can subscribe to the stream by awaiting for it. And then once　new data does come through, we can get some code to be triggered and we can get access to that new data.

Now coming back to Flutter land　however, we need to convert this data that we're getting back from these streams into actual widgets.

In other words, we need to find a way of taking the data inside our Firebase collection and display it　inside some widgets in our chat screen.

So how can we accomplish this?

Well let's tackle this in two steps.

Step one is just displaying the data in the simplest possible way.

And step two will be to make it look pretty.

And in this lesson we're going to focus on step one, just getting the data to display.

And I reckon the simplest possible way to show our chat messages is to show them all inside a column.

So before doing anything fancy, I'm simply going to display my chat messages by creating a column with　a list of text widgets as its children.

Here's what our app looks like right now　and here's what we're aiming for.

This is what our screen with that column of text widgets will look like by the end of the lesson. But　here's the rub,　our Dart snapshots method gives us a stream.

So we need something that can handle a stream and will create a list of text widgets for us.

But it doesn't just need to create our widgets, it needs to update our widgets every time a new chat　message comes into the stream.

So how can we do that?

Well we can use the help of a widget called a stream builder.

Now this is something that will turn our snapshots of data into actual widgets　every time new data comes through. So it's able to rebuild every time that there's new data coming from　the stream,　and it does that using the set state.

So in other words, set state will be called every time there's a new value in the stream. So down here　inside our column,　we currently only have one child inside the list of children. So let's go ahead and add another one.

And because we've set our column to have the main axis as spaceBetween, if we have more than one item　inside here, then it'll push this send text field and button down to the bottom and we'll get our messages　showing up at the top. And we're going to show those messages using a stream builder. Here's a stream　builder and we can use it just as we use any other widget.

So it goes into our column and our stream builder has two properties that are really important.

One is the stream, where is the data gonna come from?

That's gonna be the same as what we did up here.

You can either copy this part of the stream or you can just type it out from scratch.

So we're gonna be using our firestore.collection tapping into the messages collection and then　we're going to fetch these snapshots which is a stream. In particular, it's a stream of query snapshots.

The query snapshot is a class from Firebase which will ultimately contain the chat messages that we're　after. Now that we've got the stream that we're subscribing our stream builder for,　so it now knows when new data comes in to rebuild itself.

Well the next thing that we have to provide is a builder.

The builder is a strange one.

If you read the quick docs, you see that we have to provide something called a build strategy.

In other words, we have to provide the logic for what the stream builder should actually do.

Why?

Well remember the stream builder is interacting with our stream and with each new event like a chat　message being sent, our stream builder receives a snapshot. At this point the builder function needs to　update the list of messages displayed on the screen.

In other words the builder needs to rebuild all the children of the stream builder namely the column　of text widgets that I mentioned before.

So what is this snapshot that you can see here in the quick docs?

We can see that the type is async snapshot.

This async snapshot represents the most recent interaction with the stream. Our chat messages are buried　somewhere in this async snapshot and we can get access to it through the builder function.

So let's take a closer look at our builder. And a builder is something that takes an anonymous callback　and it has two inputs. So it's going to trigger the callback passing in the context and also the snapshot,　then it returns an actual widget.

Remember in our case, the widget that the builder function will return is going to be our column containing　our text widgets. Let's go ahead and create that anonymous function, first of which is going to be the　context that we're going to pass in.

And the second thing is going to be that snapshot that we get back. Now inside these curly braces,　we should have access to both the built context as well as the snapshot where the stream builder lives.

This snapshot is not the same as the snapshot we had over here up here.

We were dealing with Firebase's query snapshot.

But down here we've got Flutter's async snapshot because we're working with our Stream builder.

However, our a async snapshot actually contains our query snapshot from Firebase.

Let me show you what I mean.

Let's start to dig the chat messages out of our async snapshot.

So the first thing I want to check before I use it to render a widget is I have to make sure that we

didn't get an empty snapshot.

So we can check by tapping into the snapshot and calling .hasData.

So this returns a boolean telling you whether or not this snapshot has data or it's null.

And if it does have data, then we're going to use that data.

So we're going to create a final variable called messages and we're going to set it to equal snapshot.data.

This is how we can access the data inside our async,snapshot.　and you can see this in the documentation for async snapshot. In the documentation for async snapshot,　you can see both the data property that we're accessing right now and the hasData property that we're　using to check that the snapshot from our stream is not equal to null, our query snapshot from firebase　is stored inside our data property.

However if we look back at our code in Android Studio, we don't see this query snapshot data type.

Instead, we see that the data type is dynamic. And the reason is because even though we built this stream　builder, we didn't tell it what type of stream we're going to get.

Whereas if we take a look at our snapshot, we know that it's going to return a stream of query snapshots.

So this is a data type that comes from Firebase cloud Firestore　and it's something that is going to represent all the data in our messages collection. So we can add　that type to our Stream builder. So we can tell it that we're building a stream builder and we're listening　on query snapshots.

That is the data that's ultimately going to be inside the stream.

And now once, we tell it what kind of data it has,　well now the data object gets updated to an actual query snapshot which means that it now knows what　other things it has such as its documents that it will contain.

Let's pause for a second to consider what we just did here.

The async snapshot contains a query snapshot from Firebase.

We access the query snapshot through the data property.

Now we're dealing with a query snapshot object so we can use the query snapshot's properties like the　documents property.

And this will give us a list of document snapshots.

So in summary, our async snapshot from the stream builder contains a query snapshot from Firebase. The　query snapshot in turn　contains a list of documents snapshots. With two layers deep right now,　but we have to dig even deeper to get to the text of our chat messages. So how do we get hold of a single

document that contains the message text?

Well right now, we've got a list of documents. For that, we use our trusty old friend, the FOR loop. And　I'm going to use a FOR loop to build a whole bunch of text widgets.

So let's create a list of text widgets. I'm gonna call it　messageWidgets and I'm going to set it to a empty list to begin with.

Now the next part is I'm going to use a for in loop.

So for var message in messages. So that's our list of documents snapshots. Inside this FOR loop,　I'm going to tap into the message text by tapping into message.data and here data is what we printed　out before　and this is a map of keys which are consisting of string and the value can be dynamic.

So this is the data contained inside a single document snapshot.

Remember each document contains two pieces of information, our chat message text and our sender.

Now looking back at our code, it's worth quickly pointing out that even though we see this .data property　in two places, here and here, we're dealing with two very different types of objects.

They're both just happened to have a property called data. The snapshot is an async snapshot from Flutter　and the message is a documents snapshot from Firebase.

And if we take a look inside our console you can see that each of these are a piece of snapshot data.

And the key is this, whatever we called our field, and the value is what it contained,　so the text. So the message text is saved in our database under a field called text.

So we're going to type that out as the key as a string and you have to make sure that you have exactly　the same here as you do right here.

So there's no typos and no change in capitalization.

The other variable that we need to pull out of our message snapshot is the message sender. And this is　going to be equal to the message.data　and we're going to tap into the value under the sender key.

So now that we have message text and message sender, then we're finally going to create our message widget　which is going to be a text widget that contains some data namely the message text and the message is　going to be from the message sender,

so messageSender. And now we're ready to add our message widget to our list of message widgets by using　the add method.

So once the FOR loop is done running, then it's finally time to actually return a widget as the output　of our stream builder.

And I'm going to return a column because it's going to contain a whole bunch of children and the children　of the column are going to be our list of text widgets, each with a message and a sender.

So that's what I'm going to put in here.

And now once we close that off with a semicolon, we can hit save. And you'll be able to see all of the　messages that we've created thus far inside our chat screen.

And because we've got our MainAxisAlignment for the main column as spaceBetween, then it pushes that　text field down to the bottom and it pushes our column of text widget right to the top.

Now if at this point I send a new message to our Firebase Cloud Firestore, then the moment that it hits　my database, we're also going to be notified that a new message has come through and it gets added straight　in to my chat screen.

Why does it get added straight to the chat screen?

Well because our stream builder receives a new async snapshot, this will trigger the builder function　which will rebuild the list of text widgets on our screen.

So in other words, our code is reacting to a new event like a chat message being sent.

Just let that sink in for a minute.

I think this is a really really neat way of doing things.

But if we take a look in our code, the thing that we haven't really handled is, what if the snapshot has　no data?

Well in that case, what do we output for our Stream builder?

Let's think about this for a second.

In which case might we have no data in our snapshot?

Well we might have a null value in our very first snapshot before we had a chance to connect to Firebase.

So let's handle this situation.

If our first snapshot has no data and we're not yet connected to Firebase, then what I'd like to do is　to show some kind of a spinner or progress indicator, that way our users know to wait for our app. So　instead of checking whether if the snapshot has data and doing these things, I'm actually going to flip　it on its head. So I'm gonna find the closing brace for the snapshot has data and I'm going to close　it off up in the same line and I'm going to check to see instead whether if the snapshot has no data.

So this way we can check to see if we didn't get any data back yet,　then in this case we're going to return a different widget.

We're going to return a center widget and inside the center widget, we're going to use a CircularProgress　Indicator.

Now a circular progress indicator is what our modal progress HUD was based on.

Remember we implemented a progress spinner after login and registration in a previous lesson.

The modal progress HUD allowed us to easily specify when it should spin and when it should stop spinning　through the use of that boolean that we called showSpinner.

But in this case, what we actually need is just a simple circular progress Indicator.

Maybe we'll give it a background color or we'll change it to maybe a light blue accent, but this is only　here for the case when we actually have no data yet because as soon as we get data, our stream builder is　actually going to build our column instead.

So this is only going to show and spin when we have no data and only for that moment.

And then as soon as we get data it will get destroyed. Now　at the moment, we've got quite an ugly list of messages showing up. and it doesn't look like a great chat　app.

Nobody's really going to use something that looks like this.

So in the next lesson, we're going to jazz it up make it look like a real messaging app by making message　bubbles and also learning about Flutter list views.

So for all of that and more,　I'll see you on the next lesson.