Now in the last lesson we used a stream builder widget to be able to update our Flutter app with the

messages that get sent and any new messages that come through. In this lesson, we're going to focus on

improving the styling and user experience of our chat app.

What does this involve?

Well let me show you what we're aiming for first.

Here's how our app will look

by the end of this lesson.

As you can see the individual chat messages are nicely separated out in these beautiful blue bubbles.

But that's not all.

We'll actually be able to scroll through our chat messages and also when we write a new message and press

the send button,

our text field will be cleared.

Now let's see where we are right now.

What do you think will happen for instance when we have too many messages to fit into the space on our

screen?

I want to show you firstly what happens if we have too many messages and instead of creating 20 more

messages, I'm simply just going to change the styling of our text widget.

So I'm going to add a style property and inside this text style, I'm going to change the font size from

what it currently is, which is probably about 14 pixels, to let's say 50.

Let's make it a lot bigger and see what happens.

Now if we take a look at our app, you'll see that each of the text widgets are going to become gigantic.

And because we have lots of messages, they're now flowing off the screen.

And even if I drop the keyboard you can see it's still going off the screen and it's so large in fact

that it's pushed the send button and also the text field off the screen entirely.

Now this is not what we want because we can't have a chat app that only allows you to receive like five

messages

right?

So let's change this.

What we would want instead is something that scrolls so that we can have infinite, almost infinite amounts

of data inside our chat screen and we only need to see one part at a time.

So we scroll through it to see older messages.

Now in order to do this in Flutter, we have something called a ListView. And it's remarkably simple to

use especially when you consider the alternatives of how much effort it takes to set up a list view

on Android or a table view in iOS.

Let me show you all that we need to do. At the moment,

our Stream builder is returning a column with children which is a list of text widgets.

Now we're going to keep the text widgets as it is but instead of using a column widget, we're going to

change it to a ListView widget.

And because we also have our container inside the same column which contains our send button and also

our text field,

we're going to wrap the list view inside an expanded widget. This way it won't try to take over the whole

screen and only take up as much space as is available.

So now if we hit save and we take a look at the app, you can see that we now have our messages in a scroll-

able listview and we can scroll it to the top and the bottom.

And we've also got our text field and send button here.

So now if I raise the virtual keyboard, you'll see that my list view is still there in the background,

it doesn't really change and we can still scroll through it but my text view now pops up and down pending

on the size of the keyboard.

So now that we've implemented the scroll behavior, it's time to start styling our chat screen to make

it look more like a real chat app.

This is another good opportunity to see Flutter's hot reload in action and how it can really help with

our workflow.

Now the first thing I want to do is I'm going to add some padding to my list view.

So let's add a bit of padding and it's going to be edgeInsets.symmetric.

And I'm going to add about 10 onto the horizontal and maybe 20 onto the vertical.

So let's hit save and we'll see it being pushed down a little bit from the top

and also a little bit in from the left and the right. And the next thing we're gonna do is we're going

to improve the way that our messages look

because at the moment it looks terrible.

So instead of just using a bog standard text widget, let's build something custom.

So I'm gonna go right down to the bottom and I'm going to build a new stateless widget. And I'm going

to call it a MessageBubble.

And this MessageBubble is instead of going to return the container, it's instead going to contain a

text widget and also quite a lot of other things.

So I'm going to cut that from over here and I'm going to paste it inside here.

Now that I've transplanted the text widget over here,

I need a way of being able to provide those things:

the message text and the message sender.

So I'm gonna create a final string variable which I'll call the sender and also a final string variable

which I'll call the text.

And now when the message bubbles get constructed, we'll be able to initialize these values,

the sender and the text.

So instead of having the message text, it's now just code text and instead of having the message sender,

it's now called sender.

So inside our build method, instead of returning just a bog standard text widget, we're going to wrap

it inside a material widget. Because when we have a material widget we can specify maybe some background

colors.

So let's change that to a maybe light blue accent. So I want to take a look at what it looks like while

I'm designing.

So let's go up and fix our code over here so that we can actually use it.

So previously, we had the final message widget being equal to a text widget but now it's going to be

created from a message bubble. And we're going to be passing in the sender as the messageSender and

the text as the messageText.

And while we're here and we're changing the names of things to message bubbles, let's go ahead and change

these variable names as well.

So I'm gonna change all the places where it's called messageWidget.

I'm going to go to refactor and rename and instead of calling it just a bog standard widget, it's going

to be a messageBubble. Refactor that and then over here where we have our message widgets instead of

being called a messageWidget,

I'm going to refactor it and call it a list of message bubbles and we're getting an error here because

we've specified previously that message bubbles are a data type of a list of text widgets and it's now

no longer text widgets.

It's in fact some messageBubble widgets.

So now that we've fixed all the code around here, we're ready to hit save and see what our message bubbles

look like

as it stands. And as it stands, it still doesn't look great.

It's got a blue background but each of the messages are blurring into other ones.

So let's go ahead and add a little bit of padding to separate them out. Around our material widget, I'm

going to add a padding widget and we can delete the content which initialize is a object as of Dart 2.0.

So we have an edgeInsets all so let's add maybe let's say 10 pixels of padding around each of our

message bubbles.

Also now at least we can see them kind of separate from each other which is nice but it still doesn't

look great.

So what's the biggest eyesore?

Well it's the fact that our text is 50 points

large. Let's fix that and let's change this to a more normal 15 pixel font.

And I'm also going to change the color while I'm here

to colors.white.

So let's hit save and and it looks still pretty strange. Now at the moment,

our problem is that the material that's around our text is far too close to the text.

So let's give our text a little bit of padding away from the material.

Let's wrap this inside some padding as well.

And we're going to set it to maybe let's choose a little bit of a symmetric padding and we'll set the

vertical to 10 pixels and the horizontal to maybe 20.

So just a little bit more padding around the horizontal than the vertical and we're starting to look

like messages right? It's starting to look a bit better than before.

Now one of the things that you can add to a material that you've seen in raised buttons or the floating

action buttons is an elevation.

So let's add a little bit of drop shadow to each of our messages by adding maybe 5 pixels of elevation.

And it starts looking like it's a real bubble now but it wouldn't be a bubble unless it was circular,

unless it was round.

So how can we turn this square into a round object?

Well we can tap into the border radius property.

We can set the border radius to a all round circular radius, so we can tap into the border radius and

we can specify maybe 30 pixels of circular radius.

And now it looks like actual bubbles.

So now we've got some pretty good looking bubbles but it doesn't really make sense to have these sender

details inside the message bubbles

right?

So let's try and make it separate.

Let's wrap our material inside a column so that we can add a text widget at the top of the bubbles.

And here is where our sender is going to appear and then later on we'll have the material which is the

message bubble.

So let's delete that part and only have the text inside the text widget.

So now if we hit save, you can see that we have the sender details at the top and the text bubbles right

below it.

But at the moment the sender details is taking up a bit too much attention. It's too black and it's too

large.

So let's change the style of that text widget.

So that we can drop the font size down to maybe a 12 and we can change the color to maybe a black

54, so something just a tad bit lighter.

And now you can see that the main point is these messages and the sender name is just something that's

on top of it.

At this point we've done a lot of work on the styling front. But how can we improve our look even further?

Well at the moment, our bubbles are all in the center of the screen. So why don't we go ahead and move

it to one side.

So inside this column, let's change the cross axis alignment to crossAxisAlignment.end and this

will move it to one side. And we're starting to get to the point where it looks very very much like a

real messaging app right?

I keep talking about real messaging apps like it's like Pinocchio or something.

It's like you're a real messaging app now.

Well not yet actually,

we still have to do a few more things. Since we've been adding quite a lot of code into our app, it's

again time to consider refactoring our code.

Perhaps we can reorganize our code to make our chat application a bit simpler.

For example, I reckon our chat screen state is getting a bit long and unwieldy.

So why don't we extract our stream builder into a separate widget?

So down here just above the message bubble, I'm going to create a new stateless widget and I'm going

to call it my MessagesStream.

And here I want to be able to transplant all of this code that's inside the stream builder into one

place namely this place.

So instead of returning the container, I'm going to return the stream builder.

And we're going to instead create our MessageStream right here. So now the area that we have is related

to the fact that the Firestore is something that's only available currently right here for the chat

screens state.

So why don't we go ahead and move that out of there and right at the bottom of our import statements

so that we can access it all over this file? So now that we fix that problem, our message stream no longer

has any errors and we're able to create message bubbles from all the messages that gets created inside

our app.

So if I go ahead and type a new message, 'Hey there' and hit send, you'll see it appear right here.

Now don't worry too much about the order for now.

I'll explain a little bit later on why it's not going in the order of being created.

But the last thing that I want to fix in terms of styling is that the moment when I hit the send button,

it should really clear the text field because it doesn't make sense for there to be text from something

that's already sent

right?

It should be empty, ready for the next message.

So how can we do this?

Well we can create something called a text editing controller. So let's create one for our message text

field so we'll call it messageTextController. And we're going to set it to equal a new text editing

controller.

Now once we've created this controller and make sure that there's no typos in it, then we'll be able

to use it inside our text field.

So the part where we have our text field is right here and it's got a property called a controller which

expects a text editing controller.

And guess what?

We've already made one.

It's the message text controller.

So now we have this controller which can control this text field so that when the user clicks on the

flat button and presses the send button, then we can tap into that same messageTextController which

we know controls this text field, and we can tell it to clear the text that's currently inside using

the clear method.

So now if I hit save and if I conjure up another new message and hit send, you'll see that it disappears

as soon as I press the send button.

Perfect.

So that's exactly what we want. In the next lesson,

we're going to address something else. We're going to address the fact that at the moment when the message

is coming through from different senders say angela@email.com or angela@gmail.com or any other

users,

they're all showing up in the same color, in the same format.

It's really hard to tell which are the messages that I sent in which are the ones from other people.

So that's what we'll fix in the next lesson.

For all of that and more, I'll see you there.

自動スクロール