All right.

So now that we're pretty much done with getting weather from the location data in various ways, the next step is to move on to the city screen. And you can find that under the screens folder. And we can navigate to it when the user taps on this city icon which should take us to the city screen. And that icon lives over here in this flat button where the icon is icons.location\_city. See if you can remember how we can use routes and the navigator to push the city screen on top of this current location screen. You might have to do a quick google to be able to remember the code that's required to do this. But pause the video and give that a go.

All right.

So in order to push a new screen, we have to use the navigator. And we're going to use the push method where we pass the current context over and we build a material page route.

Now this builder expects a callback,so it takes a context as the input and inside the callback we say what we want to return,so the screen that we want to push. And this is of course going to be the city screen.

And in order for our current file to recognize that we of course have to import it right at the top.

So import the city\_screen.dart file and now it should recognize what a city screen is.

And finally we can end this with a semicolon and add our commas so that dart will reformat this code for us to be easily readable.

So now let's hit save and let's check it out.

Let's go ahead and click on the city icon and it pushes us towards the next screen which as you can see is pretty simple.

All it has is a single back button which should take us back to the previous screen.

And it's also got a get weather button.

Now what we need here is a text field where the user can actually enter the city that they want to get the weather for.

So in the Flutter cookbook, there's a whole bunch of articles relating to text field. And text field is just a text input widget. So it's a widget similar to every other widget that we've been using to style our UI.

And we can use various properties such as decoration and style and border to change how it looks.

So let's do that now.

Let's create our text field widget.

So I'm going to keep my app over on the right so that I can hot reload and update my style as I edit it in my code.

So let's go into our city screen and right inside this container which currently contains null as a child, we're going to change that to contain our text field.

Now this text field, if we save, you can see it show up which is just a single line. And it's following our Dart style that we defined at the beginning for our material app.

The first thing I'm going to do is I'm going to change the decoration property. And you can see this expects a widget that's called an input decoration. An input decoration has a whole bunch of properties that we can change including things such as the content padding or has a floating placeholder or a hint text, et cetera.

And what we're going to change is we're going to change it's fill property and this will make the container filled with color instead of just a single line.

So let's set that to true, hit save.

And now we should get a little bit of filledness to our text field.

And now let's change the fill color to a white color.

And now that should go from this sort of transparent white to a fully opaque white.

And then we're going to give our field a icon as well.

And this is going to use the icon creator which is going to be Icons.location\_city.

So the same icon that we had from before is going to go into our text field right on the left to indicate what should go in here.

Now we're going to change the color of our icon to a white color to fit in with our current theme.

So colors.whites, let's hit save and now that should look more matchy matchy with the rest of our app. And then I'm going to add what's called a hint text.

So this is the placeholder text that's gonna go in here to tell the user what they should type in here.

And the hint text is going to say, 'Enter City Name'. Once we hit save you'll notice that nothing changes.

Well the reason is because hint text, by default, is white.

So we have to change the hint style to change that color. And hint style takes a text style widget as the value.

And we can change the color here to a colors.gray maybe.

Now let's hit save and we should see or hint text appear just like that.

So now we've got this sort of weird shape at the moment for our text field.

It's got a single line at the bottom and then it's kind of got a rectangle.

Let's round that off by adding a border.

So we're going to tap into the border property of our input declaration and we're going to use a outline input border.

And of course this comes from reading the documentation for that border property on the Flutter docs.

And then we're going to add a border radius which I'm going to set to a radius that is going to cover all four sides.

And it's going to be a circular radius, so radius.circular.

And the value I'm going to put in there is about a 10 for the radius. And let's add our commas and hit save.

So now it rounds off all the corners to look a bit more like an actual text field, and we can now add our border side.

So I'm going to try and get rid of that bottom line there by changing the border side to none,so BorderSide.none. So now that should get rid of that border side right here and it looks more like a normal text field now. So the very last thing I have to fix is that previously we had a dark background without this white fill color.

So when we had a white text going in here as I type it's fully visible.

But now it's no longer visible because it's the same color as the background.

So let's go into our text field and update the text style that goes in there.

So I'm going to add a text style and this is going to have a color of black,so colors.black.

Now let's hit save and you can see my text should now show up and be fully visible. Now that we've created all of these things that is pretty much going to stay constant for our text field,why don't we take all of this input decoration and move it into our constants file? So let's open up our constants file in our utilities and I'm going to add a new const variable which is going to be a kTextFieldInputDecoration and I'm going to set it to equal that input decoration that we created just now and cap it off with a semicolon.

And then going back to our city screen, I'm going to change the decoration to that kTextFieldInput

Decoration, and hopefully when we hit save nothing will have changed.

Now how do we get the value that we enter into here out of the text field? Because that's essentially what we need if we want to be able to grab the weather for the city location that the user type in.

So let's say I typed in the word London right?

How do I get that value out?

Well in the text field, there's also a property called onChanged.

Just like how our buttons have a property called onTap or onPressed or onHold,we have a onChanged for our text field. And this will trigger and pass in the value that's in the text field into this anonymous callback that we put in here.

So if we go ahead and print the value, then you'll see that every single time that the text in the text field changes, it will trigger the onChanged and it'll print out the value. So let's start off with a L, so L goes in printed. L O L O and then L N D O N. So you can see that for every single change, I'm getting the print statement to trigger and print it into my console.

Now all we have to do is to store that value and pass it to our weather model to be able to get the weather for that particular city.

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