In the last lesson we saw how we were able to capture the value that the user typed into the text field.

And we did that through listening to the onChanged property.

So whenever the text field changes, it triggers this anonymous callback and it passes in the new value which we were able to print into the console.

But instead of printing it into the console, let's store it in a property.

So let's create a new string property called cityName and it's going to start out empty.

Now when our text field changes, we're going to set the city name to equal that new value that the user typed into the text field. And now all we have to do is to try and pass the city name back when the user taps on the get weather button.

So inside these curly braces, we have to somehow pop this city screen off of the navigation stack and pass that city name value back to the previous screen.

So far we've only already seen how to pass data forward, parsing a piece of data forwards to a state object.

We can do that by parsing it to the stateful widget and then fetching it through the widget property in the state object.

But now we have to figure out, how do we pass data backwards?

Well let's first add our navigator.pop method and I'm going to hover here on this pop method to show you the documentation.

You can see that it's got two values,one is the context which is the current context and we always pass that in order to pop the screen off.

But there's also a second optional input which is a result.

And this is where we're going to add the value that we want to pass back.

So we're going to write navigator.pop parse in our current context or where we are in the widget tree,and then we're going to pass over that city name variable.

Now if you had more things to parse you could parse a map for example or even an object.

But now that we've parsed this back, where can we access it in our location screen? Well if we take a look at our push method, you can see that it actually has an optional output.

It may output a future with any sort of dynamic type and that output comes from over here.

So when we popped it, we passed something over.

Well that's going to be the output of our navigator.push method.

So we can tap into it by creating a variable called typedName and we can set that to equal the output of navigated.push.

So once the city screen gets pushed on top of the location screen, then the user types in the value for city name. And once the screen gets popped off, city name gets parsed back as the output of our typed name.

But remember, this is of course an asynchronous method because you never know when the user is going to type in the city name right?

So this is why it returns a future.

So if we need to use this typed name somewhere, a little bit later on, then we have to make this await for the result of this. And which means that we have to add async to modify our onPressed and you can add the keyword async to anonymous functions like this. Or as we've done before, to named functions and method like so. It doesn't really matter as long as you put it in front of the curly braces. Now that we've got access to this typed name variable which comes from the user input from the city screen, well we can now use it.

So let's confirm that we've actually got something back by printing out the typed name into the console.

So now let's check this out.

Let's click on hot restart so that our app starts from the beginning so that we load up the location screen.

And this is where of course our navigator.push gets created.

So now let's navigate to our city page and here I'm going to add the name of the city that I want the weather for and then I'm going to click on get weather which of course takes us back to the previous screen and it's here where we get that typed name printed into the console.

Now that we've got our typed name here, let's use it to actually fetch the current weather.

But let's first make sure that it's not null.

And we can do that by adding an IF statement checking to see if type name is not equal to null.

Well in that case, we're going to actually be able to use it.

And in order to use it, we're going to call a method in our weather

.dart. But we can't use the get location weather method because that relies on getting the weather based on the current location whereas we might be nowhere near the city that the user typed in to the city screen.

So we have to create a new method here and we'll create one and we'll call it get city weather, but it's going to work very similarly.

We're also going to be using our network helper and we're going to be parsing it a URL that's going to be based off the open weather map URL.

So let's take a look at the documentation for Open Weather Map to see how we can get the weather data based on a city name.

So the first one even is get weather by city name and you can see that this is how you would structure your API calls.

So we have to add in a parameter name called Q and afterwards we set it to equal the city that we're interested in. And then we also have to pass in the app ID which we've already registered.

So let's format our URL here. Inside get city weather,let's create a var that's called url and it's going to be equal to the openWeatherMapURL.

But then we're going to add some other parts to it.

So we're going to add a question mark and then q and then an equal sign. And afterwards is where our city name is going to come.

So for our get city weather method, we have to pass in the city name.

So let's add an input for this method and we'll call it city name.

And so that we can use it to form our URL.

So let's add our dollar sign and put in our city name at this position, and then we of course have the tag on other parameters such as the app ID.

So let's add our ampersand or our and symbol and add our app ID parameter. Or you can simply just copy this part that's in green as well.

It's exactly the same.

And make sure that you don't have any typos, and then we're going to add our constant which is our API key here as well.

And finally I still want my temperature to come back in Celsius rather than Kelvin, so I'm going to tag on the and unit equals metric at the very end. And now I can add my semicolon to finish off my URL.

So now what I have to do is very similar to what we did for get location weather. I'm going to create a new network helper object and we'll call it just network helper.

And we're going to initialize it with that url that we created right there.

Now you can either do it like this in two lines or you can use what we did down here which is simply inserting this line of code into right here.

Both ways work.

It all depends on which way you find easier to read and easier to understand.

Now once we've initialized on network helper with the url that we want to get data from, then we can call our networkHelper.getData method and we can save the result of that into a new variable called weatherData.

So, so far, very similar.

And finally we can return our weather data as the output of our get city weather method.

And of course that means we have to change this into a dynamic output, but not just any dynamic output,it's going to be a future dynamic output. Again exactly the same as before.

And now we have to mark this method as an async method because get data is asynchronous.

So in order to return it, we have to wait on this to finish,so let's add the await keyword here. And that's all we need to do to set up our get city weather method to be able to fetch the weather data based off a city name that we pass in.

So now let's head back to our location screen and inside this part where we've already checked to make sure that type name definitely has a value,well now we can pass over typed name to our weather model which remember, resides in a object called weather.

But the data type is our weather model which comes from the weather.dart. And we're going to tap into that method we created just now which is get city weather and we're going to pass in the city name by using that typed name that we got back from our text field. Once we've gotten the weather data back, then we're going to save it inside a variable which we'll call weather data. And because this is an asynchronous method, remember it returns a future, then we should also mark this with an await keyword because the very next step we're going to pass this weather data to updateUI method so that we can update the user interface of our location screen.

And then that will go inside here and as long as weather data is not equal to null, then it will try to get the values of our temperature, weather icon, weather message and city name from it.

So now let's stop our app and run it from cold. And now let's head over to the city screen and enter a new city name.

Let's try Beijing maybe and click on get weather. So it pops off the city screen and it takes us back to the location screen and it tells us the weather currently in Beijing.

Now we can of course still click on our location icon to get the weather for our current location but we can also change the city to any place that we can think of.

Let's see what it's like in Paris at the moment. Cool.

So the final thing that we just need to add is this button currently doesn't do anything yet.

And what it should be is more like a cancel button. Just so that when we get over to this screen and we decide that actually I don't really know which city I want the weather for, we should be able to head back and cancel the action.

So all that button needs to do is to pop the screen off without passing anything over. So let's head over to where that button lives which is on the city screen, and it's right here in the flat button.

So at the moment, onPressed is empty but we're going to change it to navigator.pop and we're just going to pass over the context and nothing else.

Remember that if at this point if we didn't check if type name is equal to null, then we could potentially get into some problems by trying to get the weather on a null value. But because we're making that check then hopefully we shouldn't be affected.

So let's run up again and let's check it. So now if I click on the back button, it takes me back here without changing the city.

And if I click on this button, it gets me the location data.

If I click on here, it gets me the city data and my app is now working beautifully.

So I hope you enjoyed making this weather app with me.

And there's definitely a lot of improvements that you could make to this app.

For example instead of just getting the current weather for our location, what about getting some forecasted weather?

What about adding more widgets to display more values on screen? And you can check out all the cool things that you can do with the Open Weather Map APIs such as getting an hourly forecast or some historical data or even things such as UV index and air pollution.

So the world really is your oyster and I'll leave it up to you to customize and update your weather app if you want to.

But that's all from me for this module and I look forward to seeing you on the next one.