So in the last lesson we updated our location screen so that it will display the actual data that we're getting back from Open Weather Map.

That includes the temperature, the current weather condition and also a message about that city that we're fetching the data from. Now in this lesson, I want you to notice that up at the top left corner we've got this little location icon here.

And the idea is that if we decide to walk around or we go somewhere else and we want to check out the weather for the current location, then we should be able to trigger a new location request and fetch the weather for that location.

But at the moment, all of our location work is done inside our loading screen. And that doesn't really make much sense.

We want to be able to call this from anywhere within our app, not just keep it inside loading screen.

So let's do some refactoring and let's move all of this into its own file. Because all of this is all related to getting weather data.

Let's move it into our weather.dart.

So theoretically, I should be able to remove the location.dart and networking.dart and put it into our weather.dart file instead.

So right at the top, it's going to go here. And now I should be able to refactor my code so that I get rid of any of the errors. I'm going to remove the first part of get location data where I get the actual current location and I use the network helper to get the weather data and then I have this weather data object to be able to send to any screen that needs it.

So let's cut that from that method.

And inside our weather.dart, let's create a new method inside this class. And I'm going to call it getLocationWeather. And I'm going to paste what I copied over into here.

So the first thing to notice is that we no longer know what the API key is.

And that's because that's right up here so let's take that along with us and put it into our weather.dart file where it makes more sense that we have an API key for fetching the weather. Now the other thing I want to reformat is this long URL. So everything before that first parameter, everything before the question mark I'm going to cut out and I'm going to add it into a constant and I'm gonna call it the openWeather Map URL and I'm gonna put it in here as a separate string.

So now in my networkHelper, I'm gonna pass along the value of that Open Weather Map URL, and then we're going to add our first parameter, the latitude and then the longitude and then the app ID and finally the units we chose for the results is in metric.

So now the only errors I have are regarding the await and it's because we have to mark this method as async to be able to use await on all of these methods.

And finally at the very end, once I've actually gotten my weather data,

I want to be able to output it as a result of this method.

So at the very end, I'm going to return the weather data and because weather data is going to depend on a number of asynchronous methods, then this method also has to be marked to return a future and the data type of the future is going to be of the same type as the weather data which is dynamic.

So it's going to be a future dynamic object. And now we have a separate method inside our WeatherModel that deals with getting the weather for a particular location.

And when we call this method, we can get the weather data as a future output.

So now let's go back to our loading screen and make sure that we can still get access to the weather data.

So inside here, we're going to create a new weather data model.

So we're going to have to import that file, the services/weather.dart and we're going to create a new weather model which is called weatherModel equals WeatherModel and then we're going to use this weather model to get the location weather. And the output of this method is gonna be saved in a variable and we'll call it weatherData. And then that weather data is passed over to the location screen when we initialize it through this line of code.

Now if all of this is a bit too wordy for you, you can also simply just initialize the weather model in line. So we can cut that out and paste it in here because we're only really going to use it once here.

And finally we just have to make sure that we mark this method with await because we know that get location weather returns a future, right?

So this could complete at any time.

But we need to use the result of this inside weather data because we're going to pass it over to the location screen.

So unless this has a value, then this is not going to work.

Now finally when we go back to our location screen and we find our flat button which has the location icon,so that's this one right here,when it gets pressed, we should also trigger a request to get the current weather for the current location.

So inside here, we're also going to create a new variable that is a weatherData object and we're going to call that get location weather from the location screen also. Because it's now in a separate file and it's been modularised.

We can use it in our loading screen and we can use it in our location screen.

So here we already have a instance of our weather model right up here.

We called it weather. So we can tap into it over here without having to reinitialize it.

So we'll say weather.getLocationWeather and the result is going to be saved inside a weather data object. And then we're again going to call updateUI and we're going to pass in the weather data from this call.

Now remember that get location weather actually returns a future.

So if we need to use this weather data in the next line of code, we need a guarantee that this is not null.

So we have to make sure that this has completed before we go on to the next line, which means we again need that await keyword.

But we can't use the await keyword unless we mark the function with async, and we can do that in the callback in the same way that we've done for all of our other places such as our other functions and methods. So now let's stop our app and run it from scratch. And I'm gonna start off by running it on the iOS simulator. And I'm going to show you how you can change the location.

Currently, my location is set to the Apple headquarters which is in Cupertino. And that's why I'm fetching the weather for Cupertino.

But also while your simulator is open in the foreground and you see simulator right here, you go into debug, location and select custom location.

So in here you can add a custom latitude and longitude, and I've put one here for London.

So now if I select OK, the location is now set to that new custom location.

But there's a weird quirk you might expect that if we click on this button, we should be getting some updated weather data. And that would be entirely the case if you're running this on a physical iPhone device.

But the iOS simulator is not so clever.

But there's a way of forcing it to update its device location and this is what you have to do even if you're developing natively on iOS.

The trick is to go into the home screen and go into Maps. And here,

if you select this location button where you locate yourself, so we're currently in Holland Park based on that custom location I added in here, and now if you go back so exit the Maps app and go back to our weather app and click on this icon again, you can see it now reads from that custom location. And it's telling me to bring a coat just in case in Kensington and it's about 10 degrees and raining, which sounds a lot like London. If we run this on the Android emulator there's a similar case.

So it usually starts out being set to Mountain View by default, which is where Google's home is.

And if you want to change the location, you can hit these three dots and change the coordinates here.

So let's go ahead and change that to 51.5 and maybe -0.2 and let's hit send and you'll do just the same thing happens on many Android emulators as well.

When you click on the location button, it won't actually update the weather for the current location.

But if you go into the Google Maps app,so if you drag this up and open up the Google Maps and click on locate, it will trigger the location of the device to update to the one that we're specifying for the emulator in here.

And now if we head back into our app Clima and we go ahead and click on this button again, then you can see now it registers that we're in London and using these updated latitude and longitude.

Now as I said before, this doesn't happen on a physical device.

If you run the app using the code that we have on a Android or an iOS device and you walk far enough to register in a new area, then once you hit that button it should update automatically.

But it's just that the emulators have certain quirks and we can work around it when we're testing it inside the emulators.

But if you're testing on a real device, then you don't have to worry about any of this.

Now this is also a good time to thoroughly test your apps both on iOS and Android, and also on physical devices.

if you have one. And if you test it thoroughly, you might realize that this certain edge cases which will crash our app.

So for example, if I go into my settings for my app in the Android emulator and I search for the location settings and I simply go ahead and just switch off location.

And now if I try to run my app on a device that has its location disabled,this is what will happen.

My app crashes because my widgets don't know what to display because all the information that they are asked to display, for example the temperature or the weather icon, well they all come from the weather data. And the weather data at the moment is equal to null, because there is no location for a device that has its location switched off.

So we're trying to get weather data for a location that doesn't exist, then we're simply getting a null value for our weather data.

So how can we catch this and prevent our app from crashing? Well inside here where we start using our weather data to supply our widgets with something to display,we can check to see if the weather data is equal to null. And if that is the case so it could be caused by many things such as for example if the Open Weather Map the server was down and we weren't getting any valid results back or if our latitude and longitude are invalid or if we accidentally put in the wrong API key or if the internet was down, many many reasons for why we might be getting a null weather data.

And the point where we need to address this is right before we start inserting values into our widgets because that is going to mess up our screen. So we can either add a whole bunch of try catches in all of the places that can fail and that's certainly one way of approaching it. Another way is simply to check right before we use this weather data where we use it inside our widget, to check if it's actually equal to null.

And in this case then there is no weather data for various reasons and we can tell the user that there is an issue.

So here if the weather data is null, then we can set the temperature to zero and the weather icon to tell the user that there is an error. Set the weather message to say 'unable to get weather data' and then finally for the city name, we're going to set that to a empty string.

Now the final thing we need to do is to add a return statement at the end of this if statement. And this tells our app that if the weather data is null, do all of these things but then exit this function or this method. And you can use this return keyword in any of Dart's functions and methods, even if it returns void.

And what this does is it will end our method prematurely and prevent it from continuing on to the next lines, which is what we know will crash our app.

Now let's run our app again, and under the same conditions where location is still disabled.

What our app will now do is it will tell the user that there's an error and it's unable to get the weather data.

Now you can make this a little bit more sophisticated by getting rid of that in keyword which comes from this text,so you could form a custom variable for it. Or you could add a pop up or an alert that gives the user this information and keep the screen blank.

But there's many ways of handling situations where the weather data might be null.

So test your app thoroughly and see if you can address this bug.

