So you can see that in the Open Weather Map documentation, they tell us that to get data based on geographic coordinates, the API call looks like this:

api.openweathermap.org

And then we simply add the latitude and longitude.

Now if we click on this link however, it actually takes us to a separate subdomain.

It takes us to samples.openweathermap.org where it's provided a couple of samples that will work and their own app ID.

And this is so that we can see some sample data before we register on their service to get our own app ID.

But in this lesson, we're going to learn how we can use our latitude and longitude that we're getting back from our geolocator to get the actual current live weather data for that location.

So to do that, we first have to register for the API.

So go ahead and click on to the API tab and sign up for a new account. And once you've done that, you can sign in to Open Weather Map.

And once you've signed in, you should be looking at something like this.

Now here under this second list of tabs, you've got something called API keys, and that's where you're gonna go to grab your own API key. So go ahead and copy that key.

And we're going to paste it into our project.

So right off the top of the loading screen, we're going to create a new constant because this is never going to change.

We are going to call it apiKey and we're going to set it to equal that string that we copied over just now.

So now we're able to substitute this app ID with our own API key. So we can add a $ and add our API key in here. And then we're going to change the URL from samples to API to get the actual live data and we're going to change our latitude and longitude to the actual latitude and longitude

that we're getting back from our geolocator.

So inside our state class, \_LoadingScreenState,we're going to create a new variable of type double that's going to be the latitude.

And we're also going to create another one for the longitude.

And now we're going to save this data. Instead of printing it into the console,

we're going to save it into those new variables.

So latitude is going to equal location.latitude and longitude is going to equal location.longitude.

So now that we have access to latitude and longitude, we can put them into our URL.

So the latitude is here. So after the equal sign and before the ampersand, we're going to add our dollar sign and our latitude variable.

And then the longitude instead of 139 is going to be the current longitude.

So just check to make sure that you've got those values matching, so lat equals latitude and long equals longitude.

And now before we hit run, we're going to move the call to get data out of our build method which we know happens after the init state.

Instead we're going to add it right here into our getLocation. And we're going to make sure that we've actually got the current location before we call getData.

This way we don't end up parsing over some empty latitude and longitude values.

Now let's rerun our app and let's check the console. And we can see that my current value that's detected by my emulator is in cupertino and has the temperature of 285K and the weather condition is the value 803,so perfect. Our code is now able to take the current live latitude and longitude of the device and get the corresponding weather data for that location through the use of the Open Weather Map API. And then we're able to pass the result that we get back from a JSON into various values that we want including the temperature, the condition code and the city name.

Now all we have to do is to tidy up our code and refactor it so it looks a little bit better than the way it does now. Instead of doing all of the networking and requesting and response checking inside our loading screen, I'm going to split it off into its custom networking.dart service file.

So I'm going to move over both the dart:convert and also the http Dart imports.

So I'm going to cut it from here and I'm gonna paste it into networking.dart. And I'm going to create a new class here called NetworkHelper.

And when this class is initialized, so in the constructor of the class, I'm going to pass over a URL.

So I'm going to have a final variable which is going to be a string,and it's gonna be called url And when our class initializes, I'm going to give it a value for that url. Now I'm gonna create a new method called getData.

And again it's gonna be asynchronous because we need to wait for the networking to complete before we can work on the data that we get back. So we're gonna move over all of this code where we're using the http get and then where we're checking our status code and wherewe're also decoding our data.

So I'm gonna cut that out from here and I'm gonna paste it into this new getData function.

Then we're also going to include the else parts of the statement.

I'm going to cut that as well and we're going to paste it in here.

So now we're saying get data asynchronously and we're going to wait for the http get request to be made to this URL. And we're gonna check to see if the response is 200 in which case we're going to save the response body as a variable called data. And then we're going to decode that data using jsonDecode.

So now we're getting some errors because it doesn't know the latitude and longitude and the API key.

So we're going to cut out all of this and instead we're going to use this url property that we get passed over.

So inside this get method, we're going to use that url. Now inside our loading screen,I'm going to combine getting location and getting data into one method. I'm going to call it geLlocation Data. And inside here we're going to initialize a new network helper. And of course for it to recognize this we have to import that file. So import the services folder and the networking.dart file. And now it should recognize what a NetworkHelper is.

And I'm going to call it a networkHelper and it's going to be created from that constructor where we have to pass in the URL.

So I'm going to just paste that URL that I copied over from earlier, where it takes the latitude from our geolocator, the longitude from our geolocator and the app key from our constant up here.

So now I can delete this getData method and instead move everything that I've got in there to this

getLocationData. And I can delete that extra curly brace. I'm going to delete also all my print statements and I'm going to tap into that weather data we get back by creating a new variable called weatherData and it's going to be set to equal await for the networkHelper to get the data. So it tells us that we can only await on futures.

So we have to change this getData method to return instead, of a void, to return a future.

So this way we can perform this asynchronous function,wait for it to happen and then we're going to return this decoded data as the output of our method. So we can either structure it like this or for simplicity's sake we can simply just return our JSON decoded data. Now we're going to move all of these parts where we're taking out particular things that we need from our decoded data and I'm going to paste that into our location screen.

I'm simply going to put it right at the bottom of the file and I'm going to comment it out for use a little bit later on when we actually need to take these values and put it into our widgets here.

So now our loading screen is a lot simpler and all it's doing is simply using the geoLocation package to get the latitude and longitude.

And then we put that into our URL and we finally use our network helper to get the data by networking with the Open Weather Map URL that we provided here, and we check to see what the status code is and return the data that we get back parsing it using that JSON decoder.

And we no longer need to call getData and getLocationData.

We can combine it into one single call right here.

So now all we have to do is pass this data over to our location screen so that we can format it using these lines of code and put the result into our text widgets.

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