To get started with the weather app go to the GitHub repository listed in the course resources. Then clone the repository and open the project from Android Studio,just as we've done with the previous apps.

Finally remember to click on get dependencies after the project is open so that all your errors go away and your project builds successfully.

Then we should see that we have a number of folders inside our lib folder. So I've classified all of our different files into different sections.

We've got our different screens and we've got three of those: a loading screen, a location screen and a screen for getting the weather based on the city name.

And then we've got some services such as the location.dart or the networking part or the weather part. And we finally got some utilities which currently just have a constants file. And everything starts out from the main.dart file.

So go ahead and have a look at the designs that we've already created for you.

But essentially the UI is kept pretty simple because we're going to be doing a lot of backend stuff and networking.

So we've kept the user interface relatively easy to understand, even the most complex screen which is the location screen is pretty simple if you just read through it.

It's simply a container with a background image with a column inside that contains a number of buttons as well as some text widgets.

So it's pretty simple stuff so far. Now the first thing that happens when our app loads is it will load up the loading screen as the first route.

So inside our loading screen, you can see that there's pretty much nothing in here other than a state-ful widget that contains a single raised button which says get location.

So if you run this as it is, there's nothing terribly exciting going on right now,it's just a black screen with a single button. But this is where our app is going to start from and the first thing that we want to do is to be able to grab the user's location when we tap on this button get location.

Now in order to do this, we're going to leverage the power of Flutter packages. And there's a package called geolocator that we're going to be using to be able to get location services both from iOS and Android. So we'll have to call a single line of code like this that they've described in the documentation and the package will go behind the scenes and do all the necessary things to be able to get us the location whether if our code is being run on iOS or Android.

So let's get started by adding this as a dependency.

So I'm just going to copy this and go into my pubspec.yaml and right over here just below our cupertino\_icons, I'm going to add our geolocator and make sure that it's formatted to be at the same level as Cupertino icons.

And at the moment it's coming in as a string, I'm just going to delete it.

It doesn't really matter if you keep it or not.

It will be interpreted in the same way.

But I like to have things consistent.

So now that we've added that dependency, let's click on packages get to actually imported into our project.

And once that's complete, we should be able to see it inside our external libraries and Flutter plugins.

So you can see geolocator right there.

So now we can actually use it and we're going to be using it inside our loading screen for now.

So the first step towards using any of these packages is of course importing it.

So right at the top, we're going to import our geolocator.

There it is, geolocator.dart.

And now we're going to create a new method which is not going to return anything, but all it's going to do is it's going to get the location, so get location. And inside the curly braces of this method,it's going to do everything that's necessary to be able to bring us the current user location.

So if we take a look at the docs, it tells us to query the current location of the device.

Simply make a call to the getCurrentPosition method.

And it shows us how we do this.

So it's data type position and this is something called position. And then we have a keyword called a await, and we use the geolocator to get the current position.

And as an input we specify how accurate we want that location to be.

So let's go ahead and copy this line of code into our project and I'm going to paste it right inside this new method I created called getLocation.

Now the first thing that we have to do in order to make this actually work is to add a modifier keyword to this function and it's going to be the word async.

Now what exactly is this async and await?

Well this is a way for us to be able to carry out time consuming tasks such as getting the GPS location from the phone. You can imagine that asking the phone for the current location, getting it back then processing it and finally being able to have access to it takes a little bit of time.

And there's also other things such as trying to get data downloaded from the Internet or trying to read something from file.

These are all things that can take an unpredictable amount of time but also it can take quite a long time. By using asynchronous programming we can get these time consuming tasks to happen in the background instead of happening in the foreground and blocking up our UI and freezing up our app.

Now the entire next lesson is dedicated to asynchronous programming in Dart and we'll talk about async await futures and all that in a lot of detail.

But for now let's try and get the current location first.

So we've got this object called Geolocator which comes from the geolocator package and we're going to use it to get the current position by providing a desired amount of accuracy.

And I'm going to change that from high to low. And you can see that there's a whole bunch of different options you have, lowest, medium, low, high best for navigation and the thing to remember is that the more accurate the location you're trying to get, the more battery intensive it will be.

So if you want to be kind to your users and their phones battery, only use as much accuracy as you need.

So if you're making a navigation app where people actually need turn by turn directions, then you probably do need to choose something that is quite accurate like best for navigation.

But if we're only trying to get the weather, then something like low which gives us the accuracy to the nearest kilometer on iOA and nearest 500 meters on Android, is probably good enough. And even the lowest one is probably fine, but either way choose one of these accuracy levels.

And now we should hopefully be able to get our position back and once we do get our position back, I'm simply going to print it into the console to see where we currently are according to the GPS. Now there's just one more step that you have to do for this to work and that is to ask the user for permission to get their current location because you don't want apps that just happens at the user's location without them knowing cause that's kind of dangerous. And both on iOS and

Android,you have to explicitly request positional data.

So let's see how we can do that. In the documentation for geolocator, if you scroll down a little bit they'll tell you that in order to get permissions on Android you need to add either the access coarse location or the access fine location into the Android manifest,so where we're only using a low level accuracy positional data, so we only need permission to get coarse location.

So let's go ahead and hit copy on this particular tag and let's go into our Android project folder right here and then go into our app folder, our source, main and here we should find our AndroidManifest.xml file.

So if you double click on this to open it up you can see that these are various settings we have set for our Android project.

And it tells us that we have to add one of the following two lines as direct children of the manifest tag.

So what does that mean?

Well this is the manifest tag and it's the highest level tag in this particular file.

Everything in it are its children but there's also lower hierarchy such as the application tag or the activity tag or the meta data tag.

And if you've ever worked with HTML, then this formatting will be very familiar to you because it's styled in XML which stands for extensible markup language.

And essentially it's kind of the same as our key value pairs where the key are in these tags and the values are held inside the tags.

So inside the manifest tag,so right here just after the closing angle bracket, we're going to paste in our user's permission tag.

And this one tells our Android app that we need to access the coarse location and it should request it from the user.

So that's our Android part done, so we can close up our Android folder.

And now we have to ask for permission on iOS,so when an iOS user uses our app. And what we have to do is we're going to copy these two lines, a key and a string, which tells the user why we need to use their location.

So you can change this text if you wish to, if you want to make it more descriptive for the user when you're actually requesting for their data.

But let's copy these two lines and we're going to add it to our info.plist file.

So now we have to go into our iOS folder and inside our runner folder are all the files that are specifically related to our iOS app.

And here you should find a file called info.plist.

So if you double click on that you can see again this is an XML file and we're going to add the key and the string to this file just below where it says dict or dictionary.

So we're going to pay those two lines in and now we have a dictionary of a key and a value pair.

So this tells our iOS app that when we request for the user location, this is the description that we're going to give the user to tell them why we need their location.

Now let's hit save.

Let's close down those two files.

And the last thing we have to do is to actually call this method, get location. And we're going to call it when the user presses on that button that says get location,so right here in the onPressed. And in here I'm going to call get location and now we're ready to test our app.

And before you test it, I recommend just stopping the current app from running and then starting it from cold because we added in our dependencies and we messed around with the Android manifest and the iOS plist.

It's usually a good idea once you've done that to just start it from scratch. It takes a little bit longer but there's a lot of things that has to link up behind the scenes.

So now that our app is up and running when I click on this button get location, it should call that get location method I created this now and it should fetch our current position and print it out into the console.

But the first thing it will do when I click on the button is to ask me for permission to give it my current location.

So make sure that you click allow at this step. And now it should print out my latitude and longitude into the console. Now where does that location come from?

Well if you're using a simulator, it usually comes from a default location that set to the Apple headquarters,so somewhere in cupertino. If you're running your app on Android then you can click on these three dots right at the bottom here.

And this gets you into the extended controls and you can select on the location tab to set the latitude and longitude you want this phone to report from.

You can manually set and send the location data to your app.

Now at this point, it's worth mentioning that if you are running your phone on Android and you're seeing some errors in the console relating to something about Android X, then be sure to head into the course resources where there's a link to this resource on the Flutter documentation talking about how to make your app AndroidX compatible.

Now this issue won't affect everybody.

It's only if your app actually crashes and it refuses to build.

So that was how easy it is to actually get the current GPS location.

And if you think about it, behind the scenes a lot of things had to happen.

The iOS counterpart had to talk to the iOS operating system to be able to get the location from that device and the Android part had to talk to the Android OS and get the location in a completely different way.

But we were able to do that by simply using the geolocator package which has taken care of all of that for us. And we didn't really have to do anything other than simply add it into our project.

And if you're curious how geolocator manages to do this then you can head into the external libraries folder and go into your Flutter plugins and expand the geolocator library.

Now here you'll see an iOS and an Android project.

If you expand the iOS project and go into the classes, you can see that there's a whole bunch of files in here that are written in Swift to work with the iOS location API to be able to grab the location data and to be able to return it to you.

Now similarly if you have a look inside the Android folder and you go into src.main and then go into the Java folder, there's both a data folder and a task and a utils folder.

And it's here where there's all this Java code that's been written to be it to deal with the Android location manager and to be able to fetch that location data for you.

Now you can imagine writing all of this Java code and all of this Swift code not only takes a lot of expertise but also takes a lot of time. But because the author of this plugin has very kindly packaged it into a really simple to use plugin, we're able to piggyback on all of that hard work simply by importing it and calling just one line of code.

And hopefully when you build some custom functionality that you think other people might find helpful you'll also create a package around it so that you can share it with other Flutter developers and pay it forward.

But for now, I want to focus on these two keywords that we've never seen before, async and await. In order to understand this we have to learn all about synchronous and asynchronous programming.

So we have to understand what's actually going on here in this code.

head over to the next lesson and let's learn all about asynchronous programming.