All right.

So now that we've pretty much completed all of the user interface, we're ready to add some functionality

to our app.

And you'll notice that currently there's one warning that we still have that we haven't gotten rid of.

And it's related to our checkbox.

And when we try to use it inside our simulator or our emulator, notice that absolutely nothing happens.

It doesn't have any functionality yet.

In order to understand how to make the checkbox work, we first have to understand a little bit more about

state.

So what exactly is state?

Well you can think of it as the values of all the variables that together make up the user interface.

So for example, in our Todoey app at the moment, we have a whole bunch of empty checkboxes. And those

checkboxes have a property called value and that value determines what you see in the checkbox.

So if it's equal to false, then we see an empty checkbox.

But if it's equal to true then we'll see a checked checkbox.

But this is very manual

right?

So how can we do this automatically?

Well in that case we would need to have a variable which can change.

Let's say we create a variable called isChecked and we set it to false by default, and then we set each

of these checkboxes to use this variable as the state of the checkbox. If isChecked equals false, then

all the checkboxes are empty. But when the user taps on one of the checkboxes and I change that variables

value to true, then it should update the user interface and show us the latest version of our app based

on that updated state of the checkbox.

Now generally we tend to think of two types of state.

There's local state, which is information that while useful is say for example the current page the user

is on if you have an app that allows pagination but it's generally not information that we need elsewhere

within the app.

So it's not a piece of data or a variable that we would need to save its value to be used across the

app.

So in local state, it's pretty easy to use a set state to toggle the value of a variable.

Say in this case the user tapped on one of the checkboxes and we use set state to change isChecked

from false to true.

Then as soon as that happens then our checkbox is redrawn and we get a little tick because the state

of the checkbox has changed.

We've set it to something new.

Now we can toggle this by changing is checked back and forth.

But in this case we're not going to be saving this value of isChecked for use anywhere else in the

app.

So currently our checkbox doesn't have a lot of code in it.

It's a pretty simple checkbox but let's maybe add one property.

Let's change the active color to light blue accent to be consistent with the rest of the app. And then let's

go ahead and extract this checkbox as a separate widget.

So I'm going to call this a TaskCheckbox to differentiate it from the bog standard checkbox and now

that I've extracted it, I'm gonna go ahead and actually delete the constructor and I'm gonna convert

from a stateless widget to a stateful widget because I am now going to use my set state inside this

checkbox to be able to update its state. And its state is whether if the value of the checkbox should be

false or whether if it should be true. And I want that value to be based on a user input.

So in this case the checkbox has a property called onChanged and it will send the new value of the checkbox

over into this callback and we can actually create, up here, a boolean variable which we'll call isChecked.

And we can maybe, by default, actually set it to false and then down here when the user does in fact change

the checkbox which means flipping it from false to true, then we're going to change the isChecked to

equal that new value.

So now we can change the state of the checkbox to use this variable isChecked.

And then finally, we're going to add the set state that we know and love so that we can call the build

method once we change this isChecked variable to the new value that the user selected. That way it'll

build a brand new CheckBox using the latest value of the state of the checkbox.

So now if I hit save, I'm able to check and uncheck each of my checkboxes separately. So that's pretty

much local state because it's kept very much under wraps within the same stateful widget. It's very

low down in the widget tree so that the rebuilds are not very costly and we're not moving the value

of isChecked anywhere else. So what about global states?

Well when you're managing global state, you're essentially keeping the values of some of your variables

and you're using it across different parts of your app.

For example, we can represent our app as a very simple widget tree.

We have a list screen at the top which branches off into an avatar screen as well as a task list.

The task list is populated with task tiles and each of the tiles have a checkbox and a piece of text

which is the title of the task.

Now if I only needed that information within the checkbox widget, then I can just use a stateful widget

and use set state to rebuild my checkbox whenever the state changes.

But if I needed the value of isChecked somewhere else within my app, say for example if I wanted to

maybe put a strikethrough through the title of my task that got checked off, so that not only does my

checkbox update its State but also the text widget updates its state using the value of the same variable,

namely this isChecked variable.

So how could I achieve this? Let's start out by moving this variable isChecked up a level into the task

tile widget.

So let's make some space over here and then I'm just going to drag it over to here. And now you'll see

that the task tile widget is giving us a warning because this is currently a stateless widget.

So let's convert it into a stateful widget so that we can actually have a property that actually varies.

Now that we've actually moved this isChecked property up here,

we're getting some errors down here in our checkbox. If we have a stateful widget that's managing what

kind of checkbox to render as well as what kind of text widget to create,

then we don't actually need this checkbox widget to be a stateful widget. We can actually convert it

back into a stateless widget and keep it relatively simple.

So now if I go ahead and add a boolean that is gonna be a final boolean

so it's going to stay constant but every time we change the isChecked value here, then we're going to

redraw and rebuild our checkbox using the value of this boolean.

So I'm going to call this the checkboxState and I'm going to initialize my TaskCheckbox using this

checkboxState .

Cool.

So now I can go ahead and replace all the places where I was previously using isChecked but I'm going

to leave this set state part for now. Because we now have a stateless widget, we can't set the state of

a stateless widget because it doesn't have a state.

So we're going to sort that out just in a moment but for now, we're simply turning this TaskCheckbox

into a dumber widget and all it needs to do is to rebuild itself using the value of checkboxState that

gets passed in when it's initialized. So if you remember the difference between a stateful and a state

less widget is kind of like this.

So a stateful widget you can think of as something that can change and it can vary its state depending

on the value of those variables.

So recently I've been coming across a lot of really addictive gifs where there's all sorts of tables

that seem to collapse and expand and they do it in various crazy different ways.

And we can think of that as a stateful table right?

It can change and it can update and we can change it at a moment's notice depending on what we want.

But a stateless widget is kind of just like a normal table.

And the difference between them is that this table cost something like $2,000 whereas

this table cost $200.

And the reason is because this stateful table needs a lot more material,

it's a lot more expensive to produce and it takes a lot of effort to build it.

So that's the same with our widgets.

So more precisely, it's actually not the stateful widget that's changing but it's actually the state object

that is changing. Wen we have our object as a stateless widget,

it just stays as it is

and when we need to change it, we simply just get rid of this one and build a new one.

But when we have a state object, then it has state and that state can change.

So that means the variables inside the state are usually not marked as final because we change them

to update the user interface.

And once we're done changing the properties of the state object, then we make a copy of it and that is

our stateful widget. So you can see over here that we have our stateful widget class which has very

little code in it other than to create the stateful widget from the state. And down here we have our

actual state object, the TaskTileState.

And this is where we have our variables that can change its value which is why it's not marked with

the final.

And we use the changes to update our user interface in the build method whereas in a stateless widget

all the properties need to be final.

We can't change any of these properties but if we do need to update this, then we'll initialize a brand

new version of this TaskCheckbox and we'll throw away the last one.

Now notice how our TaskCheckbox initializer is requiring a single argument,

and that's the state. So we can go in here and we can pass in that isChecked value.

It could be true or it could be false

and that gets passed in to our checkbox to be able to display a tick or an empty box. So now that we've

lifted the state up to the parent of the TaskCheckbox,

well we can now access this property both inside our text widget as well as passing it over to our task

checkbox.

So that means that we can do things like adding a text style to our text widget and the one that I want

to add is the decoration.

I want to be able to cross out my text when my task is checked off, so the decoration I'm gonna use is

a TextDecoration.lineThrough.

And if I rerun my app right now, remember every time you switch a stateful widget into a stateless

or vice versa, you'll need to actually rerun it from start. Hot reload is not gonna be enough.

Now you can see that I've got all three text widgets with a line through it but this is obviously not

how I want my task app to be.

I want the decoration to be dependent upon the value of this property,

that's why we need it over here.

So instead of simply just adding my decoration in here, I'm going to check it to see whether if isChecked

is true and if so, then I'm going to add that line through decoration.

But if it's not true, then I'm going to set the decoration as null or no decoration.

So now if I hit save.

But if I change that property to say true, then all the tasks will be crossed out as well as all the

checkboxes ticked.

So this property now has been lifted up and the state is now being used in all of the downstream widgets,

the text and the task checkbox.

So that's all very well and good.

Let's just quickly restore that to false.

But the problem is that how do we get this user interaction back up to the parent widget? Because we've

lifted the state up which means we now have this stateless task checkbox which can't change and therefore

it can't set its own state.

So how do we get the user interaction back up into the task tiles state? Because I still want these checkboxes

to work right?

But at the moment nothing is changing.

So one of the ways that we can bring this data back up to the parent is through the use of a callback.

And we've been using callbacks pretty much left, right and center and in fact every single time we have

some sort of user interaction say the onChanged property of the checkbox or the onTap of a button

then we always put in a callback in there to wait for the user interaction and then trigger that function

only once the user has performed that particular action,

say changed the checkbox.

So if we want to create a callback, we can simply take all of the code that is in this onChanged property

and we can go ahead and bring it up to our task tile state.

So I'm going to create a new function called toggle checkbox which is going to be called checkboxCallback.

And I'm simply going to paste in what I copied over just now and I'm going to uncomment the parts which

are still commented.

And now that I've got my checkboxCallback which is to be more precise actually takes a boolean data

type and we've called it new value down here because it was the new value of the checkbox but maybe

to make it easier to understand, we can call it the checkbox states or the current checkbox states because

that's essentially what it's going to be.

It's going to be triggered over here in the onChanged and when the checkbox gets changed, then it will

call this callback and it's going to pass in the new checkboxState and then we're going to use that

value to update our boolean isChecked.

So now that I've created my checkboxCallback, I now have to use it and I'm going to use it by passing

it over to my TaskCheckbox when I create it.

So I'm going to add a final property which is gonna be of type function and let's call it toggleCheck

boxState. And then I'm going to add it inside the constructor for the TaskCheckbox so I can pass in

a function when I construct a new task checkbox and that gets saved inside here and I can now use it in the

onChanged.

So I'm going to pass that function right here in the onChanged.

So now we got a little red squiggly line here because it requires two arguments

and the second argument is a function which is going to be used as the callback here when the checkbox

gets changed.

So the name of our function is checkboxCallback and if I hit save, you'll see my errors do go away.

Now if you want this TaskCheckbox constructor to have some named parameters to make it a little bit

less error prone or easier to understand what's going on,

well then we can simply go ahead and add it here.

So we've got our checkboxState and also our toggleCheckboxState. All right.

So now we have our TaskCheckbox being created in the trailing part of our ListTile and we're passing

over the current value of the

isChecked property via this checkbox state property. And then we're also passing over our callback, our

checkbox callback. And that checkbox callback gets triggered when the checkbox changes when it will pass

over an input into this callback namely the latest state of the checkbox and then we're gonna use that to

update

the state of our property.

And so that means anywhere where that property is used namely in the text widget as well as in the task

checkbox widget, then both of these widgets are going to be rebuilt and we're going to see it update.

So let me go ahead and run my app as it is.

And now when I click check, notice how both my text widget as well as my checkbox now update their stae.

And it is all down to our checkbox callback here. Now

notice how in this case we're actually creating a named function. But normally when we use callbacks,

we're actually using them anonymously

so without the name and without the return type.

So we can do that here as well.

So if I take all of the rest of my callback and I just cut it and I replace it over here, then I can

delete all of that.

And now I have my callback which is being sent over to my TaskCheckbox right over here and used inside

the onChanged property. So we're now pretty certain that it works.

But if you want a refresher or just a little bit more in depth on how callbacks actually work in Dart,

then in the next lesson we're going to do a deep dive on that.

But if you're pretty comfortable with how the code is working so far then you can go ahead and skip

the next lesson and continue building out

our Todoey app. So for all that and more,

I'll see on the next lesson.

