So in previous lessons we've managed to lift the state of whether if our checkboxes checked and we're

now able to make our checked off tasks reflect both in the text widget as well as the checkbox widget.

So that's pretty neat.

But at the moment even though we have our user interface for our Add task screen, it doesn't yet work.

If I add something in here and I click add, well nothing happens and that is of course because in the onPressed

of this flat button, there's actually currently no code other than a single comment.

So let's go ahead and try to get this working.

The first thing we have to do is we have to make our tasks more dynamic because at the moment we're

essentially just generating three hard coded tasks.

And they all say, 'This is a task'. Not very interesting for a to do list app.

So how can we make it a little bit more dynamic?

Well firstly let's go ahead and make our task list into a stateful widget so that we can update it's

state.

And what we want to be able to do is to have something that we can track and update so that we can update

the state when that property changes.

So we would want to create a list of task objects and we might call this list tasks.

And it's going to be set to equal a whole list of tasks each with a property for the name of the task

as well as whether if that task is completed.

So let's go ahead and create a model folder in our lib, so let's go ahead and create a new package we'll

call it models and then inside models, we're going to create a new Dart file called our task.dart.

And now inside this new file, I'm going to create a brand new class which is going to be called task

(singular) and this class is going to describe how every task will be.

So it will have a string property which is going to be the name of the task and also it's going to have

a boolean property which is going to be whether that task is done.

And then when our Task object gets initialized, we need to pass over a name and whether if the property

is done although by default pretty much all new tasks are not going to be done right?

Otherwise why are we creating it? So we're gonna give it a default value by adding a equals sign and we can

set that default value to false.

So all tasks start out with the isDone property set to false.

So far so good.

Well we can also give our task class a method and this method simply toggles the done-ness of the taskc

so we'll call it toggleDone.

And in this case all it needs to do is to look inside the isDone property and set it to the opposite

of what it currently is.

So if this is equal to true then that exclamation mark is going to set it to the opposite of true which

is going to be false.

That's simple enough

and now once we've got our task model where we can use it in our task list and we've already added a

list of tasks as the date type

but we need to import that models folder and the task.dart file in order for us to be able to recognize

that Task object.

So now that we've got a list of tasks and our screen is completely breaking,

but let's not worry about that until we've completed our code inside here. And I'm going to create a

list of new task objects.

So it's just going to be empty for now and I'm going to go ahead and paste it just two more times. And

each of these tasks is gonna get a name property.

So let's say 'Buy milk' and then

'Buy eggs' and 'Buy bread'.

All right.

So there's a whole bunch of foods that's going to be bought

and now we have a list of tasks.

But how do we convert this list of tasks into a list of task tiles?

Well currently our task tiles are actually pretty intelligent things.

They are stateful widgets because they have all of this functionality that's baked into it.

But let's go ahead and make it simpler.

I'm going to delete all of these lines of code and I'm gonna change it into a stateless widget. And

now that we no longer have a stateful widget then we can't have properties that vary,

so that's why I'm getting a little warning over here.

So we have to convert this into a final property and also we can't set state any more because there

is no state to set.

So I'm going to go ahead and take out this callback and I'm just going to paste it here for now and

then instead of our task checkbox, I'm actually going to bring my checkbox widget back up and to use

it as the widget that's gonna be in the trailing part of my list tile.

Now the reason is because now that we have two stateless widgets it doesn't really make sense to separate

out the list tile and the checkbox because both the text widget and the checkbox is going to be rebuilt

every time we change the list tile.

So it makes sense to keep them together in this case.

And besides, we've already used this example to learn all about how we can lift state up and pass callbacks

in a simple way that it's easy to see on the same page.

So I'm now ready to delete this task checkbox and I now only have a bog standard Flutter checkbox but

still with the same active color and the value and an onChanged.

So I'm going to comment out my code that used to be the callback and I'm also going to comment out the

onChanged

for now. We're going to come back and address that a little bit later on.

But we are going to use the isChecked property as the value of my checkbox. And the isChecked property

is going to be initialized when we create a new task tile.

So we're going to pass in the this.isChecked and another one that we want to pass in is the title

of the list.

We don't want it to always say 'This is a task'

right?

So when we create a new list tile we want to be able to give it a task title to put in here and that

is of course going to be another final String property called taskTitle

and it's not going to have a value, similar to the isChecked property.

So they're going to be both null to begin with but then they're going to be initialized when we create

this class.

So I'm going to add the this.taskTitle in here as well.

So now we've converted our task tile to a pretty simple stateless widget and all it has to do is to

rebuild a list tile whenever we update either the isChecked property or the task title property and

that gets passed over when we construct our task tile.

So now let's head back over to our task list and add in those arguments that are required here.

So the task title will be taken from the task list,

so I'll write tasks

and I'm going to pull out the first item from my list using the square brackets zero and then I'm going

to tap into the name property of the task object and I'm going to assign that to the task title of

the task tile. The other one that I need to add is the isChecked property and this is again going to

be the zeroth item from my task and this is going to be the isDone property. And then I'm simply for

time sake going to copy this and paste it in all three task tiles but I'm of course going to update

the index for getting hold of the next items in the task. So this task tile is going to display the information

from the first or the zeroth item from my task list,

this one is going to be from the first item and this one is going to be from the second item.

Now if I go ahead and rerun my app, then you can see that it's carrying over the task names from my task

list.

But as you can imagine, if we had to add a new task tile to this list of widgets every single time we

wanted to add a new task to our list view, you then that's quite a lot of effort

and that's quite a lot of computation.

So I want to talk about another way of constructing list views in this lesson namely the list view builder.

Now we've already built list views and we kind of know how it works now.

But this list view builder is really useful when you need to build a large number of children inside the

list view because it's only going to be called for the ones which are actually visible.

So it saves resources when you have a really large list.

So if we had a to do list that was infinite or you know say it had 2,000 task tiles inside it,

well then it would actually only use up resources to build the ones that are visible on screen

saving a lot of computation and making our app a lot faster.

So how does this list view builder work?

Well the only required property in this builder is a indexed widget builder. Well an index widget builder

is basically a callback that takes two inputs, the current build context and the index of the current

item that's being built.

So let me show you how this would work in our case. Here

I've got a repetitive bunch of task tiles

and so instead of using all of this code and hurting my fingers through typing all of it, I'm gonna go

ahead and delete all of that.

Instead I'm going to tap into your ListView.builder. And then I'm gonna pass in a callback into

the item builder property.

So I'm going to add in my parentheses and my curly braces as usual with my callbacks and then inside

here it takes two inputs, the current build context and also the index.

The context is going to be a build context.

So just as we always have a build context in our build methods be it a stateful or stateless widget,

this build context basically contains the information that tells the children widget where the parent

is in the widget tree.

So the task list is inside a container which is inside an expanded widget which in turn is inside its

own column widget and then it's in the scaffold etc. So the information of where that widget lives is

in that build context property. And that is what we're going to pass in to the list builder to be able

to correctly render the child widgets and know where to slot it in the widget tree.

Now the index is the property that we're gonna use really shortly.

So the first thing that we're going to add to our item builder is we're going to tell it what we want to

create.

So we're going to return a task tile and the task tile of course takes to named parameters, the task title

as well as the isChecked property,

so we saw that earlier on when we were building our list view pretty manually. The task title is going

to be derived from our tasks list

so I'm going to happen to the task list and instead of hard coding which one I want to pull out of

that list, I'm simply going to pass in the index that comes from this callback and that matches with

the index inside the list view that we're currently building.

So if we were building this one right here, then the index passed in would be equal to zero,

if we were building this one then it would be one etc..

Now this is not enough because this is actually of type task and it doesn't match with what task title

needs to be, which is string.

So then we have to tap into the name property, so pretty similar to what we did before

but in this case it's now dynamic because we're using that index value that comes from the builder.

Now we have a list view builder which is dynamically creating task tiles for every item in our task

list.

But of course it can't know how many items we want it to build.

So in addition to having this list builder, we can also add another property which is the item count

for our builder. So we can tell it how many of these task tiles we actually want to build.

And of course we want to build as many task tiles for the number of tasks we have. So I'm simply going

to add in tasks.length to get a dynamic value

so right now it's going to be equal to three.

So the item count for our list view builder is also going to be three and that gets rid of our errors

down here where it was trying to build something that didn't have a task associated with it.

So now we've kind of done exactly the same thing but in this case our list view is now completely dynamic

and as soon as we update the state of our task list then the list view builder is going to rebuild. But

it's only going to rebuild the parts that the user can actually see, the parts on screen.

If we had a thousand tasks, then we're not going to be using up resources building a thousand items onto

the screen but instead we're only going to build as many as the user sees,

so it's way more efficient. So now we're finally ready to go back into our task tile and figure out how

to fix our checkbox because we've broken that functionality in order to focus on making the dynamic

list view builder work.

Firstly, we'll need a callback to pass this information when the checkbox gets changed back to the parent,

so in this case the parent would be the parent of the task tile which is of course the task list because

that's where it's getting created.

Here we can again use that callback that we created earlier on and we can use that inside the onChanged

method right here.

So we're going to take this method and we're going to bring it back over to our task lists.

Now I can either do that by creating a named method here or more easily

and what you'll see more often, is we can simply added in as an anonymous function. And we're going to

pass over that callback when we create the task tile.

So over here I'm going to create a new property that doesn't yet exist and I'm going to call it the

checkboxCallback and inside here I'm going to paste in that callback that I have from earlier on.

So it's going to be triggered inside the task tile where the checkbox is going to pass in the current

checkbox state which is of type boolean,

so this is optional. And we can have it or we don't have to.

And when we click on one of the checkbox then this callback is going to get triggered inside the task

tile and it's gonna pass over the latest value at which point we're going to update the task at the

current index that we're building and we're going to toggle it's done-ness.

So remember inside our task model, we had this method called toggleDone which simply changes the is

Done property to the opposite of what it used to be.

So now we're going to call that with a set state and hopefully that will trigger a rebuild of the current

task tile and we will get the updated user interface namely the crossed out text and the checkbox showing

up. But of course currently the checkbox callback property don't yet exist.

So let's copy it and let's create it inside our task tile.

So I'm going to create a final function which is called checkboxCallback and I'm also going to initialize

it when I initialize my task tile. So I can do this in a number of ways. I can either say when my checkbox

gets changed, well there's gonna be a new value that's going to be passed in to this callback. And in

that case I can call the checkboxCallback and pass in the new value.

But of course this is kind of redundant in this way because here we have a callback that takes an input

and doesn't return an output.

And over here we have an onChanged property which also tries to trigger a function that takes an input

and doesn't have a return value.

So we can actually simply just use the checkboxCallback inside the property for onChanged.

And now if we hit save and I go ahead and check off my items, then you can see that this functionality

is working exactly as it did before but the only change is that we now have a dynamic list view builder

which is ready to build as many items in our list view as we want.

Now in the next lesson we have a challenge for you because now that we have this capability of dynamically

loading up task tiles, we now want to be able to start adding items into our task list and then updating

the state of all the user interface elements so that we can update our app and add the new items below

the previous ones. So once you already head over to the next lesson and complete the challenge.

自動スクロール