All right.

So in the last lesson we looked at how we could get tasks from our task provider and read it both inside

our task screen as well as our task list.

Now in this lesson I want to look at how we can add a new task to our task provider

and that task is going to come from our add task screen.

So when a user types a new task in here and they press the add button then that data is coming in right

here inside this widget,

this part of the tree. And that data needs to go and update our task object, so we add a new task to that

list and then we want to be able to listen for changes in that provider both inside our task screen

to update the number of tasks as well as inside our task list so that we can actually add the new item

into the list and rebuild the list.

If you think you can tackle this then pause the video now and try to give that a go and complete this

functionality of making that add task screen work and seeing it inside the list as well as seeing it

update the number right here.

And once you're done having a go and come back and I'll show you my solution and we can talk through

it all right.

So in order to add a task, we of course do that inside our add task screen. And the way that it works at

the moment is we have this text field where the user types in a new piece of text which is going to

correspond to the new task and then we save that inside a local property called newTaskTitle.

And when the user is finally done changing or typing in the text field then they press the Add button

and it's in that moment where we trigger a callback that takes that value and puts it back over inside

our task screen where we call a set state and add the new task to our list of tasks.

But now our list of tasks doesn't exist and our task screen is not even stateful anymore

so we can't use set state either.

So let's see how we might do this in the provider way.

Now one way that you might think it could be done is like this. If I go ahead and delete my callback

inside my add task screen so that it becomes just a simple stateless widget being created as the

modal bottom sheet, then I can go into my add task screen and I'm going to also delete my call to that

method right here and delete all of these properties which passes over that callback.

So now the way that you might think we could do this is when the onPressed is called, we create a new

task and that task is going to have the name created from that local property which is called new task

title.

So let's put that in there.

And of course it doesn't know about task because we're not importing the model/task.dart.

So let's go ahead and do that.

So we now have created a new task based on what the user typed in and you might think that we could

simply just tap into the provider.dart and also tap into the models/task\_data.dart and

simply write inside the onPressed provider.of and the data type will be a TaskData data type.

And now we have access to our task data object where we can simply say .tasks.add and we add

that new task over here. And then finally we call the navigator.pop right here and then we pop away

this current screen.

Now this kind of sort of makes sense in theory especially this part, but in practice if we go ahead and

run this you'll notice that it doesn't actually work.

So let me go ahead and show you.

I've typed in a new task,

I press add, it correctly dismisses that screen but nothing gets added.

So what's going on right here?

Well remember how when we were going through the examples we mentioned that you can't just change the

string by simply tapping into that property data and giving it a new value by setting it to some new

value.

You have to do it through a method because we need to call notify listeners. And that is a method that

comes from the change notifier class which updates all the places which listens to the state of that particular

property so that they can redraw and rebuild. Similarly over here

if we simply just add this new task to our tasks list and without calling that notify listeners, nothing

is going to be notified,

nothing is going to happen.

Instead what we have to do is we have to create a method and we'll call it addTask where we pass over

the string of the newTaskTitle. And inside this method is where we're going to tap into our tasks and

we call the .add and we add a new task. So we can either do it in line over here or we can create

it right here.

So final task equals new task which has the name of the new task title and then we pass it in to add

this new task into our task list.

But most importantly we call notify listeners once we're done so that we update all the listeners on

the latest state of the task object that they're listening to.

So now inside on our add task screen, instead of doing all of this we would instead delete this firstly and

we would call provider.of task data and we tap into that method we created just now, add

Task and we pass in that new task title which comes in through here via our text field. So now if I go ahead

and hit save and I try to check it out again, you can see that if I add a new task, 'Go to the gym' and

click add, you can see it gets added to the list because of that important notify listeners method being

called so that everybody who's listening to the task right here as well as in the list, they all update

accordingly. It's actually quite dangerous if we simply just leave our tasks like this because it means

that sometime down the line when we forget about all of this knowledge that we've learned about provider,

we might get into the situation again where we think oh I could just say .tasks.add and add

a new task.

So in order to save ourselves from making this mistake in the future because remember that you have

to be kind to your future self

right?

So as a programmer but also in life you have to make life easier for the future version of yourself.

So let's try and figure out how we can prevent ourselves from making this mistake in the future so we

don't have to actively remember,

'Oh yeah.

I can't just tap into this list and just use .add or .remove.

I actually have to do it via a method where I notify the listeners'.

So instead of allowing this task to be accessible everywhere keeping it public, I'm going to add that

underscore in front of it to make it a private property.

So now it's only accessible within the two walls of curly braces here and I can tap into it inside my

methods and also inside my getter right here.

So both when I'm returning my task count as well as when I'm adding a task, I'm tapping into this private

version of tasks.

And now if I tried to tap into tasks from the outside, I can no longer do that.

It's no longer accessible to anybody outside.

So I am now forced to use the add task method in order to add that new task title as a string for my new

task.

So this minimizes any errors. But now there's a kind of a problem right? Because I use that tasks elsewhere

and those places are, for example, here when I'm trying to read from that task list. If that task list

is now private

well how am I supposed to read it here?

And that's why I'm getting all these errors.

Well let's go ahead and create a getter for this task.

So down here I'm going to create a new list which is going to be of type task and it's going to be public.

So it's going to be called tasks without that underscore.

And then here I'm going to create a getter by,

remember, adding the get keyword in front of the name of the property, opening a set of curly braces and

then returning the value that I'm going to compute which is simply going to be our private version of

the task.

Now you might think that this would work because now we don't have any problems in here and we can access

the tasks. But we've now kind of gotten back to the same place

right?

I can now again tap into tasks,

I can try to add value to it

and now I'm in the same problem as I had before.

So how can we make sure that we don't do this

and instead we make sure that we use our add method and safety-check this? Well instead of just using

a bog standard list in our computed property,

our task which has the getter, I'm going to use something called a unmodifiable listview. And that comes

from the Dart collections library

so that's what I'm going to import. And then here I'm going to change that type to a UnmodifiableListView.

Now this is kind of a misnomer if you will.

I don't like the way that this class is named because it makes you think of list views, as in the widget

like here where we have a ListView. But it's really not a list view widget at all.

It's in fact another version of a list data type but it's kind of a view of another list.

So this is why they're calling it a unmodifiable list view.

It's kind of like you're looking at a version of the list but you're looking at that list through a

window pane.

You can't touch it, you can't change it

and in fact if you tried to say, I don't know, inside add task, let's for example try to tap into our tasks Un

modifiableListView and we tried to call .add on it and add a new task object with a random name.

Then even though it doesn't give you an error right here which I wish it did, but it will actually give

you an error when you run it because if we hold down command and we click on this add function and look

at how it's implemented,

you can see all it does is it just throws an error.

It tells you that you can't add to an unmodifiable list because this method is not supported in an un

modifiable list.

That's the whole point

right? We're not meant to be able to modify this list.

So now I can create my unmodifiable list view and I can return a unmodifiable version of my tasks.

So now theoretically, it should mean that when I try to access this tasks directly from the outside and

I try to modify it, I should get an error somewhere down the line.

So now I can kind of be a little bit safer in my use of this task data class and I'll now be reminded

to always use the version which calls the notify listeners which is through that method addTask. Now

that we're done with our code and we're done with increasing the safety of our code,

we're ready to go ahead and hit save and check out our app.

So let's go ahead and add a new item, 'Go to the gym'.

I really need to go to the gym.

So I'm going to add that to my list

and now you can see it updates right here where we're listening inside

our task via our consumer and also it updates right here where we're listening using our provider

.of task data.

So did you manage to get it to work and did you get tripped up by that notify listeners method call?

Again if this is confusing it's worth going into a simpler version,

so going back to that state management example app where everything is super simple, super easy to understand

and messing about with the code over that just so that you can get the hang of it and write the code yourself.

Now in the next lesson, we're going to look at the next part where we check off items in our task list

without the need of a set state or a stateful widget.

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

