So now that we've done quite a bit of coding, it's a good time to do a bit of tidying up and refactor our code a little bit to make it easier for the next thing that we're going to learn and we're going to add to our code.

Now this lesson is designed as a challenge for you.

The goal of this lesson is to simplify and reorganize our existing code.

In the process you'll become very familiar with the code in the stub and also you'll review how to modularize code, how to extract widgets and how to work with a constant file. At the end of the lesson,

we'll make some finishing touches to our app's design and you'll see how easy it is to make this change

thanks to having refactored our code.

So first things first. Let's delete all the print statements that we don't really need anymore

and let's group together all the related parts of code.

Now on our welcome screen

notice that if you scroll down, we have two of these widgets, these padding widgets that enclose a material

with an elevation.

So that's going to get a little bit of drop shadow.

And they contain a material button which is what will take us to the relevant screen.

Now these two padding widgets are pretty much identical, other than the text that they contain as their

title,

what they do when they are pressed and also what color they are. These widgets exist on three of our

screens, our welcome screen, our log in screen and our registration screen.

So here's part one of the challenge.

Can you refactor these padding widgets into a separate stateless widget

and when we use it, we're going to simply pass over a color property and a onPressed function and the

text that the buttons will display?

There are quite a few steps involved as part of the challenge.

So I've written step by step instructions that you'll find in the course resources.

So pause the video and try to complete this challenge.

All right.

So here's the solution to part one of the challenge.

Now as always, the easiest way of refactoring widgets is by simply going to the Flutter outline selecting

the top level widget which contains all the subwidgets that we want to extract, and then right clicking

on it and clicking on extract widget.

Now I'm gonna call mine a RoundedButton and then we're going to click on refactor. And now we have a

rounded button as a separate stateless widget.

Now I'm going to delete the constructor that came from that extraction and we're going to create our

own constructor which is going to initialize three properties. One which is going to be a property of

type color and I'm just going to call it color spelt either the British way or the American way whichever

you prefer,

and we're also going to have a string which is going to be the title of the button.

And finally I'm going to have a function which is going to be onPressed,

what should happen when the button gets pressed. And in my constructor I'm going to construct all three

of these.

So this.title, this.color and this.onPressed.

Now given that this is a button, it kind of makes sense for the onPressed to be required

right?

So I'm going to also add that annotation called required in front of thi.onPressed so that when we

create a round button, it can't not have a onPressed.

So now that we're done with the constructor in the properties we can actually use them inside the build

method for this stateless widget.

So instead of the color, we're going to change that to the color that's going to be passed in. And instead

of the text widget with log in, we're going to change it to the title which is also going to be passed

in.

And then for the onPressed, I'm simply going to cut the current callback out of there because I'm going

to use it a little bit later on, and then use that function that gets passed in instead.

So now that I've created my rounded button, all I have to do is to use it where I needed it.

Now as I mentioned before as of Dart 2.0, we no longer have to use the new keyword or the const keyword

when we're constructing new objects,

so we can delete that. And our rounded button currently is giving us a warning because it has a onPressed

property that is required.

Let's go ahead and fill in all the properties.

So the title is going to be log in as it was before.

The color is going to be colors.lightBlueAccent and finally the onPressed is going to be that

callback that I cut out from previously which is simply the navigator to push the log in screen onto

the navigation stack.

Now I'm going to do exactly the same thing with my other padding widget here. So I'm going to delete

it all the way down to here and instead I'm going to use another rounded button.

But this one is of course going to say register instead, and it's going to have the color of blue accent

instead of light blue accent.

And when this button gets pressed instead of going to a log in screen, it should be going to the registration

screen. So that's all we need to do to refactor that.

And now the build method for our screen is now vastly simplified and there's much less nesting going

on. So now I can actually move this class into its own file.

I'm going to create a new folder or new directory that I'm gonna call components.

And inside this folder called components, I'm going to create a new Dart file which is going to be our

rounded button. And now I can paste that class I created from before and import my material library for

it to know what a stateless widget is.

And now I can import my components folder, and the file was called rounded button.

So now it knows about rounded buttons and we've now refactored this page pretty well.

So the next thing that we're going to refactor is the registration and log in screens.

Notice here that we've also got a button here.

So we're going to replace this

register button with that same rounded button.

So let's import our components folder and our rounded button file and now let's replace all of this.

So it has a blue accent color,

it has nothing currently in the onPressed and it also has a text of register.

So let's replace that with a rounded button which has a title of register, a color of blue accent and

a onPressed that currently is completely empty.

So we'll just add a empty callback in here and we'll deal with it a little bit later on.

Now inside the log in screen, we've also got one of these buttons right here.

So this is a light blue button with an empty onPressed and a title that says log in.

So let's delete that and replace it with a rounded button.

And of course we have to import it to be able to use it.

So now we can add our properties

so our title is going to be log in, our color is going to be colors.lightBlueAccent and onPressed

is going to be an empty callback.

So that's it, done. Refactored. OK.

So now comes part 2 of the challenge.

If you look at our log in and registration screens, you'll notice that they have these text fields.

These text fields are heavily styled by our input decoration. And the reason why the input decoration

is so long both for the text field on our registration screens as well as our log in screens is because

we're actually specifying quite a few things.

We have two text fields on both of these screens

one where the user is going to put in that email and another where they're going to enter their password.

But we're adding padding to the content that's gonna go in the field, we're giving the text fields a outline

style border and we're changing the colors of all of the sides of the borders.

And we're also specifying a slightly different appearance when the text field is focused versus when

it's enabled.

So this is what it'll look like when our app starts when the screen loads up

and this is what it will look like when it's actually being activated, when the user is about to type

something into it.

So we're simply just adding one pixel to the border so that it becomes a little bit thicker.

That's all we're doing.

Now normally we would put all of our styling inside our constants file.

So we really have a constants file which contains some constants for styling our button text or our message

text field or the message container decoration.

Now what I'd like you to do for this challenge is to extract the input decoration and put it into the

constants file.

That's where we've got our other decorations and this will allow us to simplify our registration\_screen

.dart and log\_in\_screen.dart files even further.

Again I encourage you to pause the video and refer to the step by step instructions before I show you

the solution.

All right.

So as usual, when we create a constant all we do is we take the part of the styling that is repeated

or that we want to put outside of our screen code,

so we're going to cut this input decoration out here and we're going to paste it into our constants

file.

So we're going to create a new constant

and we're going to call it kTextFieldDecoration and we're going to set it to equal that input decoration

that we copied over, change the comma to a semicolon.

And for the hintText it doesn't really make sense for it to say enter your email for every single

text field

right?

So let's leave that hintText as maybe blank or just 'Enter a value' if we don't specify any hint

text then that's the default one.

Well now when we go back to our registration screen and we set the decoration of our text field to that

kTextFieldDecoration, and of course we have to import our constants file to be able to do this.

And now what we have is a bog standard text field with that default line, enter a value.

So how do we solve this problem?

How can we have a text field with basically the same input decoration widget other than just the one

property?

Well remember how previously when we looked into themes, we were able to use a particular theme that Flutter

came with such as the dark theme and then we could use that method copyWith to say 'copy this entire

theme but make this one change.'

Well we can do the same with our input decoration as well. Instead of just using the text field decoration

as it is,

we could say kTextFieldDecoration.copyWith and the one thing that we want to change out of this

mass of things we could change, is just the hintText.

So for this top text field, the hint text is 'Enter your email' and for the bottom text field the decoration

will be kTextFieldDecoration

and we're going to use copyWith to specify a hintText which is 'Enter your password'. Now we're able

to use the same constant that defines how each text field would look including those rounded borders etc.

but we're actually specifying one change to that whole thing which is changing the hint text. And we

can do that across the log in and registration pages.

So we're going to replace that with the one that we got from our constants file.

And this is the last one that we have to replace here as well.

So now we can actually delete this a dark theme which remember makes all of our text white.

So when we go through our log in or our registration pages, there's actually no hintText being displayed.

But if we delete this dark theme and we run our app then you'll see that the light theme shows up the

text as black but it also means that the text in our text fields show up with this grey color. And you

can see that even though both text fields are using the same decoration,

so if we had no decoration,

well this is what it would look like.

So let's try cutting that out and hit save.

And you'll see that on the registration screen, we've got that bog standard sort of text field instead

of the custom one that we have here.

But if we go ahead and put that back in and hit save, then you can see it turns into that lovely rounded

outlined text field and you can see it even highlights by changing the width of the border

when we click on it or when we tap on it. One last thing that we have solved is because we've now gotten

rid of that dark theme in the beginning of our app, our buttons are now light themed.

So the text in them are black.

So in order to change this across all the buttons because we have refactored our code and we know exactly

where the code that is responsible for rendering all of these buttons on the welcome screen, on the log

in screen, where that code is located namely inside the rounded\_button.dart file, then we can simply

change the color of the text in this one place.

So inside that text widget, we'll add a style property which is going to be a TextStyle which is going to

change the color of the text to colors.white.

And when we change it here, it'll go through across all of the places that we use that button and we

don't have to mess around with searching for it through a long build method or through the code that's

on our screens.

So here are some first hand benefits of refactoring and I recommend that while you're creating your

Flutter apps and while you're coding, to do some refactoring on just a regular basis. It makes it much

easier rather than leaving it all until the end when you run out of steam and you probably won't have

the time to do any refactoring and it ends up being sort of spaghetti code.

So now that we've refactoring our code and our registration screen and log in screen now look pretty

simple and pretty straightforward,

we are now finally ready to get started with implementing Firebase and adding a back end database for

all of the messages that the users will create into our Flutter app.

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

自動スクロール