Now slowly but steadily,
we're coming to an end of this arguably really big module
but there is a lot to teach regarding Routing
as you can tell.
And this is the complete guide course after all,
not the 'I just teach you some parts of it' guide
but still we're getting closer to concluding this module.
But there still are a couple of important features offered
by React Router you should know.
And to introduce the first feature, you'll find a couple
of updated code files attached to this lecture.
You'll find an updated main navigation js file where
I added a new link and this new newsletter signup
component, which is another component
you'll find attached newsletters sign up
which renders a simple form with a input
which could be used by users
to sign up for an imaginary newsletter.
There also is a CSS file for this component attached.
Then I added the newsletter.js file
which should go into the pages folder
which also renders this newsletter signup
component wrapped into this page content component.
And where I defined dummy action.
That doesn't really do anything
but that does extract the provided email
and we could then send it to some backend server.
But here we're not doing anything with it

because we don't need to do anything with it for this demo.
And in app js, I also added a new route
and therefore you'll find the updated app js file attached.
I added this newsletter route
on the same level as my homepage route essentially.
And this route renders the newsletter page component
which I just showed you
and has the newsletter action attached to it.
Now, I did add these components
and this route because there is a new feature
I wanna show you, as mentioned.
And for that, it's important to understand
that we have this newsletter signup form,
both on the newsletter page as well as on every other page
because it is included here in the main navigation.
So it's not just on one single page, but on all pages.
The problem with that is that
of course we wanna trigger this action
this newsletterAction
whenever this newsletter form is submitted.
And if we are on the newsletter page,
if we are on this page,
that would be quite straightforward to do.
All we had to do is go to the newsletter signup component
which is where we have this form
and use React Router's form component here
which starts with a capital F.
And as you learned that would automatically

trigger the action that belongs
to the currently active route.
The problem is however, that this form is included
on all routes because it's part of the main navigation.
This newsletter signup component is part
of the main navigation.
Therefore, we would have to add the action to all routes
and that would of course be a lot
of code duplication and also clash
with other actions that we might need for our routes.
Now this is such a common use case that React
Router has a solution for it.
There is a special hook which you can import
from react-router-dom, and that is the useFetcher hook.
The name might be a bit strange
but this hook when executed gives you an object.
And this object includes a bunch
of useful properties and methods.
For example, it gives you another form component
which is different from that other form component
we used before.
It also gives you a submit function
which is different from the submit function we got
from useSubmit, which we used before.
But what is the difference between this form we get here
and this submit function which we get here?
Well, if we use this Fetcher Form component like this
which we can then this will actually still trigger an action

but it will not initialize a route transition.
So Fetcher should basically be used whenever
you wanna trigger, an action, or also a loader
with help of the load function without actually navigating
to the page to which the loader belongs
or the page to which the action belongs.
On this form here we can add the action attribute
and for example, point at /newsletter because I know
that I wanna trigger the action off that newsletter route
but I wanna make sure
that I don't load that route's component.
I don't wanna load the element that belongs to this route.
And with the default form, we would do that.
If I add form here without Fetcher
just to show you what the difference is
you will notice that if I set up this form
like this with the default form provided by React Router
if I go to Events and I then enter some email address here
I'm forwarded to the Events page after submitting this.
And that's not the behavior I want.
Now, it changes if I use fetcher.Form
because as I mentioned with Fetcher, we don't transition,
we don't move to a different route.
So now we can get rid
of the form import and use Fetcher form instead.
And now if I'm on Events and I enter my details here
now I submit this without transitioning.
Now we don't get a feedback

because I haven't added any logic for this
but we saw before that it seemed to work
since we did transition with that other form.
And we can get a feedback by using other properties provided
by Fetcher because useFetcher,
this useFetcher hook, is basically the tool you should use
if you wanna interact
with some action or a loader without transitioning.
So if you wanna send your requests behind the scenes,
so to say, without triggering any route changes
and because that's the goal
or that's where Fetcher wants to help you
this Fetcher object also includes a bunch
of properties that help you understand
whether your action or loader that you triggered succeeded.
You also get access to any data returned
by that loader or action.
You get access to that data
through that data property here to be precise.
So we can actually use object Destructuring to pull
out that data property, that data object, which is returned
by the action or loader that's being triggered.
And you can also get hold
of a state object or a state value to be precise
which is equal to idle loading or submitting
which you might know from the useNavigation hook.
But useNavigation was meant to be used
with actual route transitions.

The state you get from Fetcher instead tells you
whether the Fetcher behind the scenes completed its loader
or action that was triggered.
So we can then use this to update the UI accordingly.
For example, we could bring back
useEffect the regular good old useEffect from React
and trigger a function whenever data and state changed.
So whenever one of these two values changed
and we can check if state is equal to idle,
which means we're not executing an action
or a loader anymore, and we can check if we got data and
if that data got a message property because actually
my newsletter action does return an object
with a message property.
So that's what I'm checking for here.
And if that's all the case, we could use the built
in alert function to say "Signup successful"
or anything like that.
We can actually also just output data.message.
That might be even easier.
We could of course also do other things and for example
clear the input or do whatever we wanna do.
But this is all the way of getting access to the data
and the state of our behind the scenes action
or loader execution, and they offer
with those changes applied.
If I now submit this, again, I get this alert here.
So useFetcher is the tool you should use if you

wanna trigger a loader or an action

without actually loading the page,

the route to which this

action or loader belongs.

And it's perfect for scenarios like we have it

here where you might have some shared component

or a component that's used multiple times

on the same page and where you just wanna update

or get some data behind the scenes.