Section 14 blog

Are at my friends are all down with our blog application in this video.

I want to do a quick wrap up on some of the really high points of some of the things that we learned

inside this app.

So let's get to it.

Inside of a root index such as file we had imported redux thunk and then we imported.

Are we wired up to a redux store through the use of Applied middleware right here.

We called Applied middleware which is a function from the redux library itself.

We passed the result of that into the second argument of the create store call when we apply that middleware

of redux thunk.

Anytime that we dispatch an action the action will be first sent to redux thunk as the middleware and

then after redux thunked the action will be sent off to all of our different reducers when we wired

up redux thunk it changed the rules of our action creators.

So inside of our action creators file we no longer had to create action creators that always returned

in action object instead with redux thunk.

We could also optionally return a function if we return a function.

It would be automatically called with the dispatch and get state arguments and that essentially gave

us total control over changing or getting information out of our redux store.

Anytime that we expect to make an API request from an actual creator we are always going to make use

of something like redux stunk.

There are other options out there to allow you to make API requests or otherwise async behaviors but

redux thunk is by far the most popular one.

When we return a function from our action creators we use this really interesting syntax right here

and remember all that's going on right there is we have a function that returns a function like so.

So we had the outer function outer function and then whenever we called it it automatically returned

this inner function just as you see right there.

The syntax that you see inside of here is very common.

You're going to use it on quite a bit of projects any time that you make use of redux sunk.

Now for the alternate solution to solving over fetching stuff we saw how we could create an action creator

that somehow called other action creators and made sure that we still dispatch the result of calling

those action creators.

All right so the last thing really quickly are reduced.

So we learned a ton about our reducers.

We learned that the first argument is what I refer to as states and it is whatever was returned from

this reducer or the last time that it ran so we run these reducers over and over again and whatever

whenever we return shows up as that first argument.

We learned that we usually make use of this switch statement syntax inside a very douceur.

Again this is a extremely common pattern.

Inside of our user's reducer we saw some other really common syntax that we're going to see in a lot

of reducers.

So remember anytime that we return some data from a reducer we always have to return a new array or

a new object or a different valued string or a different value number if we expect redux to realize

that we made a change to the data inside of our application.

If we ever just return the exact same object or array redux has that very simple comparison where it

just checks to see if that is the same object or array in memory.

And if it is redux says oh no data has changed and it does not update the rest of your application and

tell the Riak side of your app to actually re render itself and pull down new state and show some new

content on the screen.

OK so that's a quick highlight of some of the big topics that we discussed inside of this application.

So we still have a lot more to learn.

So let's take a pause right here.

We'll start talking about the next application we're going to work on in just a little bit.