Working With Lists In React

Video Transcript

Video 1 – Introduction To ToDo List And CRUD

A ToDo list is a great way to explore the functionality of a front-end framework. And because of it, you're going to see it a few different times within the course, we will use it in different contexts. In this case, we're going to look at the pure ToDo list tasks that are pretty close to CRUD, Create, Read, Update, and Delete. That is part of most applications that we write. We'll start off with some basic concerns. Where do we keep state? One of the most important decisions when it comes to designing an application. Who are the parents? Who are the children? What is the separation of concerns?

We will then add some styling because to be realistic, that is always a consideration when we're talking about a front-end application. We will then look at how do we capture data from the user, that is the form that we write. We will introduce this component to our list. And then ultimately, the R within CRUD, which is how do we remove an item? Once we have a set of code that is managing state the way we want it. That is being styled. That has the input form and that has the remove capability or functionality. Then we'll take a step back and ask the question, how can we refactor our code?

How can we write this in a more transparent way? How can we group our functionality so that we're composing at a higher level of abstraction instead of having spaghetti set of concerns within our code? Now, once we do that, we'll see that there are once again, a number of options as to how do you group your functionality, and how do you, where do you keep your state? So, as mentioned, this is a good example when it comes to to-do list, to think through the number of concerns that go into creating an application, designing your components, making decisions on where to keep state, how to style, and so on. And so, because of it, it's a good one to think through.

Video 2 – ToDo List Overview

Making a ToDo list turns out to be a good way to introduce yourself to the concepts of a framework. And this is what we will do here. We will start off with a classic ToDo list. As you can see there, I have a list of ToDos, and also an additional field for a new one. Now, we will break down this into a number of steps to not have lessons that are too long. And the very first one is going to be the list and also the state for these items. We will follow that up by looking at how we can create a form that will add to this state, that will add to the list. Then we'll add some styles, as this is always an important part of an application. We will then look at how to remove one of these items, how to remove something from state-based on click event.

And then finally, we will look at how to refactor the application so that it becomes easier to read, easier to maintain. And it just looks better and is easier to understand, it's easier to digest when it comes to picking it up for a first time. And that is important, that is maintainability. This is something that you always have to consider when you're creating applications. And so, with that in mind, we're going to go ahead and do the following lessons. As I mentioned, we'll start off with listing. Then we'll add some styles, then the forms. We'll then remove ToDos, and handle the other cases that we mentioned. All in all, we will look at seven lessons on how to build a basic ToDo list.

Video 3 – Set State And List ToDos

In this lesson, we will start our ToDo application. We will set the state and list our ToDos. So, let's go ahead and move to the editor and get started. As you can see here, I have an empty editor and I have a blank page in the browser. I have the editor pointed at a directory called 'SAMPLE', and I'm going to go ahead and add my first file. This will be 'index.html'. And inside of it, I'm simply going to paste some boilerplate for a React application. I'm going to go ahead and wrap my syntax here. And I'm going to go ahead and make some comments about what I have. This is the title of my application, and this is going to be call 'ToDos' or yeah, let's make it plural, 'ToDos'. Then, I will add the same as a header in my document. This part here is the transpiler.

The lines 18 through 20 are the React libraries. And this is the element within the page that we will be targeting. Now, everything else that we write, we will write inside of ' "index.js" '. And let's go ahead and create that file now, 'index.js'. Now, let's get started by creating the function that has our top component. We will call it 'App()'. And inside of it, we're simply going to 'return()' our JSX. And for now, I'm going to make it empty. Now, below it, I'm going to go ahead and add to the 'DOM'. And I'm going to use 'ReactDOM.render()'. And the parameter that I'm going to pass here will be my '< App/ >' component. And then I'm going to target the element within my HTML that I'm going to pass all this content into.

So, I'm going to enter 'document.getElementById'. The 'Id' of that element is '('root')'. Now, inside of my component, I'm going to go ahead and enter an empty tag, for now, just a fragment. And as you can see there, my red underline goes away. Next, let's go ahead and add our initial state. And we're going to use useState in a minute. And so, we need a variable name and we're going to call it 'todos'. And then the function that is going to allow us to set that. We will use to 'set' naming convention as usual. And so, I would call it 'setTodos'. Then I'm going to use the 'useState' feature, as I mentioned before. And inside of here, we're going to set our initial state. Now, I've gone ahead and written that beforehand, and so I'm simply going to paste it here.

That is going to be an array of objects. And I'm going to paste the array of objects. And as you can see there, I have three of them and they all have 'text:', which is the description of the ToDo. And then they have a Boolean, 'true' or 'false' here 'isCompleted'. And you can see that all of them are initially 'false'. So, let me go ahead and line up my objects here. And so, there as you can see, we have our initial state. And the next thing that we're going to do now is that we're going to

create the JSX so that we can display these in the browser. Now, inside of here, I'm simply going to add an expression. I'm going go ahead and collapse the left-hand sidebar there.

And inside of it, we're going to take a look at the 'todos', and we're going to 'map()' to that. And the parameters that I'm going to be using from the map callback signature is I'm going to call each one of those that are passed in 'todo' as opposed to 'todos', plural. I'm going to use the singular here. And then the index is going to be an 'i'. Now, the syntax that I'm using here is ES6 for functions. And I'm then going to have a tag '<>' that I'm going to be using to list each of those items. This tag will simply be a '< div >'. And I'm going to pass in a 'key' here. And the value of that key is going to be the index.

And then I'm going to have another expression here. And that expression is going to be simply the text of that todo. So, I'm going to enter '{todo.text}', as you can see there. And then I'm going to go ahead and close that tag. This will be my '< / div >'. So, as you can see there, if we review again from the top, I've created a component within which I am creating an initial state, which is a number of todos. Each of those 'todos' are initialized with a value 'isCompleted' at 'false', and also has a 'text' describing their 'todo'. Then I have a 'return' where I am looping through all of those 'todos'. And then I am creating a '< div >' tag for each one of them, where the 'key' is the index value and the 'text' that set inside of the '< div >'.

So, that would be the inner text, would be to do that text. So, now with that, let's go ahead and save our file. Let's go ahead and fire up our web server, and then load it into the browser. And I've just started my server at port 8080. So, I'm going to go ahead and enter that. And as you can see there, we get our list of 'ToDos'. We get the one across the top, which is the one from our HTML page. Nothing else because everything else we are constructing is inside of 'index.js'. And then we have a number of '< div >'s, as you can see here, that match each of those items inside of that array that we created initially within the page.

And so, you can see there that we have 'learn react', then 'meet friend for lunch', and 'build todo app'. And so, in summary, what we did is we created a component. Inside of the component, we created an initial state. We then created within our return JSX a loop to create a '< div >' for each of those items in the todos. And then we can see them here be enlisted within the HTML page. So, now it's your turn. Replicate what I have. Add some additional todos, and make sure you understand the cycle and the rendering of that component.

Video 4 – Styling The ToDo List

In this lesson, we will now be changing the logic, but we will be adding styles. We will add some custom styles and we will add them to the tag that is listing the ToDos. So, let's go ahead and move to the editor. Here we will start out by adding a file. This file will be styles.css. And inside of it, I'm going to paste some styles that I have written beforehand. As you can see here, I am defining a number of properties, and a number of classes. You can take a screenshot if you want to have the same style. However, you can keep it style minimally, or you can do some other type

of styling of your own choosing. I'm going to go ahead and scroll here to give you a chance to see all of it.

And as mentioned, you can take a screenshot if you want to have the exact same styles. So, I'm going to go ahead and save that file. Next, I'm going to go ahead and add that style files to our application. And I'm going to do it inside of my HTML page. And I will enter here 'link' and use autocomplete to fill in the rest. And since I have it in the same directory, I'm simply going to list the name of the file, which is ' "styles.css" '. And I'm going to go ahead and save it. I'm now going to use some of that styling within the tag for our element that we're using to render our ToDo list. And I'm going to add here 'className='.

the class that I'm going to add is going to be ' "todo" '. So, let's go ahead and save that page, reload the browser. And as you can see there, things look a lot better. We have some nicer font. We have a list there that looks like a list. And so, we've added at this point some minimal styling, but that's all we have done. So, we had the same thing, same logic as before. However, now we are styling the list. So, now it's your turn. As mentioned before, you can go ahead and use the same styles that I did. Write some of your own or import some from some other styles framework of your preference.

Video 5 – Input Form - Add ToDo Item

In this lesson, we're going to be taking user input, that is adding ToDos to our list. We will do that with a form. So, we will add a form to our code. And then we will handle the submit event to be able to add it ToDo to our state in our application. So, let's go ahead and move to the editor. And let's get started by creating one more variable. That will be a manage variable. That is, it will be in state. So, I'm going to go ahead and enter 'value'. That is the input we will take from the user. And we will set it by using 'setValue'. Then I'm going to enter 'React.useState', and I will initialize it as an empty string. Next, let's go ahead and write our form. I'm going to go ahead and enter my tags here. And inside of my form, I will add an 'input' element.

And I'm going to add a number of attributes. The first one is going to be 'type="text" '. The next one is going to be the 'className'. This is how I will style this input field. And the class that I'm going to apply is ' "input" '. Then my 'value' is going to be the value that we just defined within the state of the application. So, this is simply going to be '{value}'. Then I'm going to add a 'placeholder' to give the user some indication as to what to do in this field. And we will simply enter here, "Add ToDo ...". And the last attribute is going to be the 'onChange' event. And we will use 'onChange' event to set the value in our input. And so, we're going to enter an expression.

This will take the event. And then we're going to set our value. And we're going to take the value that was entered. And we can access that by entering the event, '(e.target.value)'. So, that will give us the input. Now, we need to have an onsubmit event on my, our form to be able to map that to a function where we can take that value and then add it to our list of ToDos. And so, I will enter here, 'onSubmit'. And then the name of the function that we're going to write shortly is going

to be 'handleSubmit'. And so, now let's go ahead and write our 'handleSubmit' function. We will use the ES6 notation. So, I will enter 'handleSubmit'.

Then use the arrow notation. We're going to go ahead and take the event. Then I'm going to define the body of the function. The very first thing that we're going to do is that we're going to prevent the default. By default, the behavior is to reload the page, and we don't want that to take place. So, we will simply enter, prevent that by entering 'preventDefault();'. Next, we will check the value of the field. And if the '(!value)' is empty, then we will 'return;', meaning do nothing. Otherwise, we can go ahead and construct our new ToDos. And I'm going to go ahead and do this here on a line on its own that is simply going to calculate or evaluate the 'newTodos'. And we will get access to the current list. And then we're going to add to that the new value.

So, it will be 'text:'. And the 'text:' property is going to be the value that has been entered. And then 'isCompleted' is going to be initialized to 'false' because whenever a todo is first created, it has not yet been done. So, that's a good assumption. Then we're going to go ahead and set our Todos to the '(newTodos)'. So, we're changing, we're going from the old state or the current state to the new state. Then we will clear out our form, which is 'setValue(' ')' to be empty once again. So, before we go any further, let's review what we have done. We went ahead and added a new variable and that variable is a manage variable. Stating the state of the application, which is called 'value'.

And we set that value using 'setValue'. Then we created a form that has an input element where the user can enter that value. When that form is submitted, we handle that event with 'handleSubmit', which is the function we just wrote. Inside of it, we prevent the default, which is to refresh the page. Then we check for an existing value. Then we construct our new list, which is the existing list plus the new Todo. We set that using 'setTodos' and then we clear out the form. So, let's go ahead and save the file, reload the page. And as you can see there, we have the placeholder for 'Add Todo ...'.

And I'm going to go ahead and here, simply enter 'one', and I'm going to go ahead and hit 'return'. And as you can see there, we have a new item on the list. So, let's go ahead and enter something more, more like a Todo item. 'go running'. And so, we could go ahead and add as many as we wanted to. And you can see there that the state is being updated. So, with that in mind, now let's go ahead and give you a chance to write some code. Go ahead and take a look at what we have done. Walkthrough creating the form for input. Make sure you understand all of the events that are being raised.

How the value, for example, is being updated by using our 'onChange' event. And that when we submit the function that we handle it with submit. You understand the event that is being passed, and what is in that event. You can put a breakpoint in your debugger and take a look at all of the information that is passed to that function. And make sure you understand how everything there takes place. And then how do we update the state? We add one more item. We create a new array with a number of objects, and then we update the state with those new Todos. Very last step that we do, is 'setValue'. So, now it's your turn. Go ahead and give it a try.

Video 6 – Remove ToDo Item

In this lesson, we're going to add functionality to remove items from our ToDo list. We will do so by catching the onclick event. That is, if you click on one of the list items, we will catch that event and then remove one of the items and update the state of our ToDo list. So, let's go ahead and move to the editor and get started. To start, we will add an additional attribute to our < div >, which is used to list our ToDos. That will be an 'id='. And the value of that 'id=' is simply going to be the index. That is the position within the ToDo list of data object that we are listing the specific ToDo. Then we're going to add our 'onClick=' event. And so, 'onClick', we are going to call the function '{removeTodo}' which we will write next.

And so, that's '{removeTodo}'. Now, let's go ahead and write our 'removeTodo' function. I'm going to go ahead and use ES6 syntax here. So, that's going to be 'removeTodo'. I'm going to use 'e' to denote the event that will be passed to this function. Then inside the body of the function, I'm going to start off by getting a handle on that id that is going to be part of that < div >. So, I'm going to go ahead and start here with a 'const' of 'index'. Then I want to make sure that the value that I'm being passed is in fact a 'Number()'. And this is a JavaScript function that will make a number from a string if that is in fact the string. And so, in this case, we're going to access the value using '(e.target.'.

So, the event 'target.id)' is the value that we set here for the < div >. That will give us the index position of that ToDo, the one that we want to remove. And so, to be splicit, let's go ahead and update the state in a couple of lines as opposed to doing it all in one. And first, I'm going to get a handle on the current '[...todos];'. This is the current list. And then this temporary list I'm going to update by removing that ToDo, and I will remove that using the 'index'. And so, I'm going to pass the '(index,' and pass '1);', meaning to remove simply that one item from the list. And then we're going to set our ToDos all in one step. So, we're going from the current state to the new state. That is the '(temp);' variable.

And now we have updated it. So, we have committed fully from the old one to the new one without any intermediate steps. And you can see there each of those steps. So, when you're debugging, you can put breakpoints if you'd like to be able to see how this list is being updated and then set the new value or the new list. Now, let's go ahead and save that page. And then we're going to go ahead and reload the page in the browser. And now let's go ahead and click on an item and let's see if we can get the expected functionality. And as you can see there that is gone. We can go ahead and add more things to the list and we can remove them as well.

Now, it's your turn. Make sure you understand how the id was set here. And then the event of onClick was handled and mapped to the 'removeTodo' function. Make sure you understand as well what was passed on that event, what 'e' holds, and how we were able to access the id, go ahead and dig in there, put a breakpoint, take a look at everything that is pass, all of the information that you can count on whenever you pass an event onto a function. And then how we updated the state using 'setTodos'.

Video 7 – Refactor - Form Component

At this point, we have the functionality we want, we are able to list our ToDos, we're able to add to that list, we're able to remove items. However, our code is not as clean as it could be. We have everything in one page. We are not separating when it comes to potential components. And so, our code is a bit messy. And being able to break off pieces and put them into components is one of the ways in which we can clean up our code. And it turns out to be great for reusability as well. And so, let's go ahead and get started by refactoring our code, breaking out the part that is the form functionality. And so, let's go ahead and move to the editor and do that. So, let's get started by creating a new file.

We'll call it form.js. And inside of it, we will copy over some of the functionality that is specific to the form. But let's go ahead and get started by writing our 'function' to hold the 'ToDoForm( )' component. And for now, we will leave the parameters empty. We will simply enter a 'return;' here. Now, let's take a look at our index.js, and let's see what type of things are specific to the form. One of the first ones is '[value,'. '[value,' is used for the purposes of taking input from the user, which is what the purpose of the form is. So, we can take that from index.js, and we will add it to our form. Next, let's take a look at the form tag, within the page. You can see here that all of this is specific, once again, just to the form.

So, we can take this away. And we can add it here to the 'return', going to go ahead and put in the line 'return'. And then I'm going to go ahead and paste the code that I have from our index.js. So, we've taken out those two pieces. Now, we need to be able to handle the 'onSubmit' event. And for that, we can go ahead and take the 'handleSubmit'. Now, let's take a look at that function and see if we have any dependencies on some of the state that was being handled in index.js that is not part of that form. And if we go down the list, the first line is fine. The second one is too. But then when it comes to being able to access ToDos, this part is not. So, we can handle that by passing a function.

That function, let's say it's called '({addTodo})'. And in this case, we can simply 'addTodo', and set, and pass that '(value);'. However, that function needs to exist within index.js. So, let's go ahead and write that next. We will write it here. I will use the ES6 notation once again, 'addTodo'. Then that is going to take as a parameter 'text', which will be the value from the form. And then in the body of the function, we're simply going to paste those two lines that we removed, which are getting access to the local state and then updating the list as we did previously. Now, at this point, we have everything we need. We just need to add that new component, which is going to be the form.

And in this case, I've called it 'TodoForm'. And what I'm going to go ahead and add here is an attribute for that 'Todo' function. And I'm going to go ahead and enter '{addTodo}', which is the name of the function. Then I'm going to go ahead and close that tag. Go ahead and save my file. Go ahead and save form.js. And then I'm going to go ahead and reload the page. And as you can see there, I have an error. And that is because 'TodoForm' is not defined. And that is because I

have not yet added it to our file that is doing the loading of all of the other files that we're using in this application, and that is index.html.

So, I'm going to go ahead and add 'form' here. Then I'm going to go ahead and reload the page. And as you can see there, we get the listing as expected, then the 'TodoForm'. Let's go ahead and see if that works, and that is not working. Let's take a look at our console, and it says value is not defined. Let's go ahead and take a look at index. And I suspect this is the problem. Let's go ahead and reload the page. No error. Let's go ahead and add one. Yep, that works now. Let's go ahead and click it and we can remove it as well.

So, as you can see there, the code is much cleaner since we've removed all of the functionality of the form as well as the form tags, and we have broken it out into a separate file. So, at this point, now go ahead and try remove that functionality. Think about, how the state is being managed. Think about how that function is being passed. Think about the handling of the state. Think about who's the parent who's child, and try with some changes yourself. Try to make some additional changes and try to refactor your code so that it's cleaner and easier to read.

Video 8 – Refactor – ToDo Component

In this lesson, we're going to continue our refactoring work. We're going to once again break out a piece of functionality and put it into a component. In this case, we're going to do that for the Todo component, the one that is listing each of the items on our Todo list. So, let's go ahead and move to the editor. The first thing that we're going to do is to create the Todo file. So, that's 'todo.js'. As before, we're going to create our code for the component. This is going to be called 'Todo()'. For now, it will take no parameters. And then inside of it, we will write the body. And to get started with the body, Let's go ahead and take a look at 'index.js' as we have in the past. And let's go ahead and take the code for 'Todo' and paste it here into 'return'.

Let's go ahead and wrap our code. Now, as we take a look at our code, we can see that there is some information that we don't have now that we are on a separate component. We don't have the index. We don't have the function, and we don't have todo. So, this is information that would need to be passed in. And I'm going to list those parameters now. So, a '{todo, index, remove}', and 'remove' being the name of the function that will be used to remove one of the items for him 'todo'. So, I'm going to go ahead and remove these two since we will no longer use them. And I'm going to go ahead and call the function here simply to differentiate it from the one that we have in the parent. I'm going to call it '{handle}'.

And then our '{todo.text}' is going to be the same as before. I'm going to add some, some styling here. Not really styling, but just an indicator that if we click on, if we click on that line, we will be able to remove the item as before. And now, I need to write that handle function. I'm going to go ahead and enter here 'function handle()'. Then I'm going to enter the body. And here, I'm simply going to 'remove', and pass the '(index);'. This will be the function that is dedicated to removing an item from the Todo list. So, this is a much simpler, a much simpler component than the one that we wrote for the form. So, now let's go ahead and move to index. And let's remove the old way of creating the todo items.

Let's go ahead and add the new component. And the new component is going to be 'Todo'. And we're going to go ahead and close it. Now, inside of it, we're going to pass the 'index'. And the index value here is '{i}'. Then 'todo' is going to be the value that, the value of the item on the todo list. And so, this is going to be '{todo}'. And then 'remove' is simply going to point to the remove Todo function, '{removeTodo}'. Now, since 'removeTodo' is now being a the parameter that is being passed, now comes from Todo, and what is going to be passed there is the '(index)'. We are no longer dealing with an event. And so, this will be here, simply 'text'. And actually, let's call it 'index', so it's easier to read.

Then the rest will be the same. So, we will update the array of objects just as before, except that the value now is being passed from the component. So, I'm going to go ahead and save this file, going to go ahead and reload the page. And now, we have an error. And the error is just as before. Let's go ahead and take a look at it. You can see there that 'Todo is not defined'. And that is because once again, I have not added the file for Todo. So, I'm going to go ahead and add that now. Let's go ahead and save it. Let's go ahead and reload the page. Let's go ahead and see. Oh, this is a warning because of the of leaving out the or taking out the id. In this case, if I put in 'id' back, I will get rid of that warning.

It thinks it's a list that we will act on, which we are, but we're not creating. We don't need the id, in this case. So, we go ahead and add that. Let's go ahead and reload the page. Let's go ahead and look at the list. And I've, I'm still getting the error. And the reason for that is because it's not looking for an id as you might have seen. It's looking for a key. So, let's go ahead and reload the page once again. Take a look and as you can see there the error is gone. Now, let's go ahead and take a look at what we did before. And you can see there that the functionality is still the same one. I can remove, I can add, I can list all of my items. Now, in this case, the clean-up or the refactoring can be, you could argue that it's not that different than what we had before. The code, I would say looks cleaner.

Our tags are simpler. There are dedicated tags that they indicate what their function is. So, the Todo is a Todo item. Todo form is a form for taking input for Todo. However, our code is not that different, but now it's your turn. Work through the example, see what additional changes you would make. Make sure your understanding of the parent and child relationship is the one that you think. Make sure you understand how parameters are being passed back and forth. And that, perhaps you want to do a different grouping or a different arrangement. So, go ahead and play with the design. Think about your design decisions. A lot of it, a lot of creating applications, depends on where you hold your state, on how you do your design. So, this is a good one to experiment with. So, now it's your turn.

Video 9 – Final Style Touches

Let's go ahead and finalize our application by adding some final style touches. Let's go ahead and add a couple of < divs > that will wrap the rest of the tags that we have there. We're going to go ahead and start off by adding a '< div >' where we will use the 'className' of the app. And as you will remember, all of these styles were previously defined. And if you want to see them once again, you can take a screenshot there or you could add your own. I'm going to go ahead and scroll so you can see the complete list. Now, I'm going to go ahead and go back. And here we're going to go ahead and add ' "app" '. This will be our outermost < div >. And then I'm going to go

ahead and indent this a little bit, add one more '< div>'. This one is going to be for the ToDo list. So, I'm going to go ahead and enter once again the 'className'.

And the class that we're going to add is the ' "todo-list" '. Let's go ahead and wrap our inner tags and indent. Just to clean up at this point we no longer need the fragment tags. I'm going to go ahead and save and reload the page. And as you can see there, we have a more styled application. And we can go ahead and add more entries. We can go ahead and remove them by clicking on them. And you can see there that it's sort of completes the application. It looks more like a fully built application. So, now it's your turn. Go ahead and add some styling. You can use the styles that I showed you. You can add some of your own that you create from scratch. Or you can use some from the many style frameworks that exists like Bootstrap. This is the end of this application. I hope you have enjoyed it. Go ahead and finalize it by giving it some cool styles.

Video 10 – Introduction To Rendering List Exercises

One of the most common things we need to do in React is to handle lists of things. So, these could be short lists like routes that we need to be taking, where we click on a button to go to a different page. Or these could be lawless. In these exercises, we're going to consider button clicks and how we handle those, how we capture them. So, if we want, for example, to delete a button, that we need to know its identity and there are various ways of handling that in React and we'll dive into those. Then also we look at how useEffect can be used and what causes useEffect to be called. So, it takes two arguments. And the second argument is really important because depending on what that second argument is, useEffect will behave differently. So, we're going to dive in, and look at does in some detail. And I think this will clear up some of the hooks that React has particularly useState and useEffect.

Video 11 – List With Simple Data

So, in this exercise, I want you to take this code and we're entering a minibar, and it's '[1, 2, 3, 4, 5];'. What I want you to do is to change this so it only prints out the 1, 3, and 5. So, I don't want any even numbers in there. So, I want you to change this so that there won't be any even numbers returned. Okay? So, you need, instead of map, I'll give you a clue. You need to use a filter to create that list and then have a printed. It should look like this to start. And then I only want 1, 3, and 5. And also, I'd like you to make these buttons.

So, let me show you what that would look like. So, that's the answer, is we want these to be 1, 3, and 5 as buttons. So, take a look at how you change that code to do that. I've given you the Bootstrap button, so understand capital Button. So, you need to change these. Okay? And notice that when they're not buttons, the unordered list now doesn't have the little full stop in front of it. This '{ { listStyleType: "none" } }', means that this unordered list, if you're using lis like this, won't have any little full stops in front and that's exactly what we saw in this case.

Video 12 – Rendering List

So, let's get some practice with React, and in the standalone environment. So, I'm going to kick up the server, 'http-server -c1 -1'. So, and it's running on port '8080'. And that means we can run our code. And here's our code. We're going to use standalone, which you've seen already. And we're using JSX. So, I've got a file called 'lists.jsx'. We're going to get some practice rendering in React. And here we're going to pretend that we're creating a nav bar. And in this case, it's just a series of numbers that we might make links to. But I want to render it. That's the first thing. So, let's take a look. So, I want to take each of these numbers and render them as an unordered list.

And so, I need to first create a list of each number and embed it between the '< li >' tag that React can understand. Notice, we're returning here something that doesn't look like pure JavaScript, but for JSX it's fine. So, what we're going to do is create something called 'updatedNums', it's just a 'const'. And later we're going to use that and embed it in an unordered list, and have it rendered by the 'ReactDOM'. So, we're going to use array '.map', 'numbers' is an array so we can use map on it. And we're taking each of the numbers and we're returning it embedded in the '< / li >' tags. So, 'updatedNums' is going to have a series of these '< li >' tags as an array.

And now, we're going to embed that in the '< ul >' list. Okay, so make sure you understand how a array'.map' works. And let's take a look at the code running. This is a unordered list. So, it renders 1, 2, 3, 4, 5 out. That's fine. That looks good. You can imagine these could be a navigation bar if we click on them, we go to some page. We've got one problem though. And it says that each child of the list needs to have a unique key property. So, React needs to identify each element that we're rendering. And in this case, they do it by using a key. So, let's go take a look at the key. Well, we'll need to insert it. We need to add a 'key' here. And we can get that if we, for example, let's add 'index' to this. So, this is, it's going to pass in both the array element.

So, '[1, 2, 3, 4, 5];' and its index. So, 0, 1, 2, 3, 4, and we're going to take the index here. And we really should make it a string. So, to do that, it will work without it being a string but it's better that we do this. Because if we change the order of these elements, somehow, we'd like to keep the same identifier against each one. So, that should correct what we have. Let's go back and take a look at, okay. And I will find this time. Let me bring up the IDE. It's fine. So, that got rid of that problem. Let's now take a look at this version. So, again, we're going to render this list of numbers, but we're going to create a component to do it, and we're going to call it 'NavBar'.

So, here we render 'NavBar', and we pass in a property, 'menuItems={menuItems}'. So, we're passing in this array, and we're picking it up here as properties. So, this is one way to get data into your component, is to pass it through the properties. So, we pick it out as 'menuitems' out of properties because that matches that name there. So, now we've got list as being '[1, 2, 3, 4, 5];'. Now, we can do our mapping again. List out will the items. We remember we need to put a 'key' in here. And we need to make this 'index'. We need to put another bracket around it, its now. Okay, that looks good. We've got that, and that should be our same unordered list. Let's save that, and re-render.

Okay, so it renders and in the IDE it's fine. We don't have any errors. So, that's rendering that. Let's, I'm going to ask you now to make a change. Why don't you make a change to make, instead of having these as allies? Let's make them Buttons And let's use Bootstrap to do it. So, we can do that and we can pick up '{Button}' from 'ReactBootstrap;'. Okay, so I want you to repeat what I'm doing now. So, I'm going to make these 'Buttons'. So, we'll just change these. Okay, and now let's take a look at it rendered. Nice. Now, we didn't make them list items, so that's why they're rendered across the page. We should take a look at our CSS. These are now buttons and could link us some onclick to a page. We'll come back and figure out how to pick up onclicks and continue with our nav bar.

Video 13 – Rendering List With Buttons And Handling onClick Events

So, we've seen how to take a list of numbers, an array of numbers, and render them out as buttons like this. Now, what I want to do is I want to know which button am I clicked on. So, let's take a look to see if we can figure out how to do that. So, we're reading in 'menuItems', we're passing into the 'Navbar' component. And now, we've rendered it by using the 'map' function to get an 'updatedList' where we have button elements around that list of numbers. Okay, so now we want to add to that button an onClick. So, 'onClick', we can call a function, I'm going to call it '{handleClick}'. So now, we need to write that function. So, here's the outline of it. And it's going to be quite simple.

We're just going to raise an alert, but we need to pick up the event. I'm putting brackets around it just to show you it'll, you don't need them. But that event, if we go to '{e.target', we can get at the '.innerHTML}'. And the 'innerHTML' is what's written on those buttons. So, here, it'll be this text or this character actually, 1, 2, 3, 4, 5. So, we can pick that out and we can then identify anything to do with that button. So, this is the handle we can get on the button. So, let's save that, and now, let's check out our code. So, I need to reload it and now, you clicked on 3. You clicked on 5. So, we know which button now we've clicked on and we can control it.

Video 14 – Anonymous onClick Function To Pass Multiple Arguments

So, we've been talking about onClick events when we click on a button. So, let's take a look at the code and at the moment in our 'NavBar', we're calling '{moveToCart}'. When the button is clicked, 'moveToCart', automatically has the event passed to it and we're digging into the event here to get hold of 'e.target', will be the name of the element. So, it's going to be this button and it can give us access to all the properties of that button. Now, one of the limitations of 'onClick' is it can only take one argument, that it can only pass one thing. And we might want to pass other things to the cart.

So, for example, it might be nice if we knew the 'id' of the product. So, we might want to past two things to the cart. Now, we can't just do it with that. Now, the solution is that we can put an anonymous function here. So, we can have the anonymous function and we can pass in the event.

So, that's the one argument that we're allowed. But now we can make a call to 'moveToCart'. And we can pass in the event, but we can also pass in, for example, an 'id:' object. So, this is the way around this limitation is that we call an anonymous function. And inside that, we call the function where we want to pass in multiple arguments.

Of course, we can pass in any number of arguments, or this could be a very complex object. And inside here, for example, then we can pick up, let's print out that '(id)'. So, let's take a look at the new code. So, let's reload that. And I'm going to bring up the IDE. So, we see here that we get hold of the id of that. So, that's the way around the limitations of the onClick is to specify anonymous functions. There are other methods, but this is the preferable one. There are some limitations if this function is being passed to children, for example. But this gets around this limitation of just one event being passed to a function.

Video 15 – Saving State With UseState

So, we've seen how to render buttons, and how to get at which button has raised an event. So, let's see, how would we record this? We might want to record the order that these buttons were checked in. Okay, let's set that as our task, and go and look at the code. So, here we've got our code when we're reading in 'menuItems' to 'NavBar'. And now, we want to record those clicks. So, here we are. This tells us which button's been clicked on. Let's see if we can record that and save it. Now, to save it, we'll need to use useState. So, let's say '{[clicks,' and 'setClicks]'. So, this will give the order that we're clicking in, which may be important. For example, if you are dialing on the telephone, you can imagine.

So, we want to 'React.useState([]);'. And we'll have an array, and we'll have it has nothing in that array. Okay, so we've got that. Now, here, Instead of 'alert', let's put that event. So, we'll store that. We will 'setClicks' to that. This will overwrite clicks. So, we don't want to do that. What we want to do is we want to spread 'clicks' the array and then we want to add this onto it. So, this should do that. This will record the clicks. Okay? How are we going to see if this works? This 'setClicks' is going to cause 'NavBar' to be re-rendered. So, every time we come in here, we can write out '(`clicks:,' and then we'll put '$ {clicks}`)'. Okay, that should do it. Let's go test that out. Yes, it's all hooked up. So, this is the piece that's new. And this is a nice way to record events so that they just have one added after another.

So, we spread this one. This isn't an array, so we don't spread it. We just add it on the end and we put it into an array. So, that's one thing that you should note. It's really useful to do that. Okay, let's go see if. Okay, we've reloaded it. Let's take a look. click: 1, 2, 3, 4, 5, 4, 3, 2, 1. Yup, so this is giving us a nice history of the state. And you can see that every time we click, we reload that component, it's been re-rendered. So, you can see that this is where things are being written out and that'll be written every time this is re-rendered. On useState, when we save the clicks when we do setClick, it's going to fire a re-render. Okay, so that's how to save our state and we'll come back and put it all together.

Video 16 – ReactDOM

So, let's take a look at when React re-renders. So, we've seen with these buttons that we can record the clicks, clicked on: 4, clicked on: 5, et cetera. Now, I put a write statement in when NavBar is first loaded, or every time it's re-rendered, it's going to do this and also this 'alert(`Rendering NavBar`);'. Now, at the moment, I'm not updating clicks. I've commented out 'setClicks' here. So, I've just got you '`clicked on:'. So, what's happening is that this will get loaded and re-rendered once. And we'll see the alert come up and it'll write this. And then every time we click, it's going to write this as well. But we'll see that it's not re-rendering. Let's look at it. So, here we are. Let's restart it. Comes up with the alert. And it comes up with clicks as being empty.

Now, let's start clicking, clicked on: 4. No more alerts but we're certainly picking up the clicks. So, it's not like everything that we do with this interface causes it to get re-rendered. But one thing that does cause it to get re-rendered is if we write into 'useState'. So, here, now when we click, I'm writing into 'useState'. Let's see how it performs now. So, let's save that, and let's go back here and let's reload. So, NavBar, we're hitting that. Now, let's see what happens when we click NavBar again and click down. But all the time. Now, it's rendering out clicks: as well. So, you can see every time we go to useState, NavBar is reloaded. Every time we change something in the Shadow DOM, then it's going to re-render everything.

And we'll take a quick look at this. So, here's one view of the Shadow DOM. It takes care of synchronizing things between the Shadow DOM and the DOM. So, here the Count is 2, Count is 2. Now, React decided to have a Shadow DOM because every time something changes in the real DOM, the browser parses the HTML to find the node. It removes the child element of this specific element. It updates the DOM and recalculates the CSS, updates the layout, and finally, traverses the tree and paints it on the screen. So, there's a lot of work in the real DOM, it's quite slow. So, what they've done in React is they keep track of their Shadow DOM and they keep track of the Old State and the New State. And if the Old State and the New State are the same, they don't bother re-rendering.

If there's a change like there is now that Count is 3 in the New State and 2 in the Old. Then they go and they'll re-render and update it. So, the New state now will get rendered in. So, if we do something by directly going to the Browser DOM, the React Shadow DOM may not be aware of it because it's just comparing its Old state with the New state. And if it hasn't changed, then they won't re-render. So, when we were clicking on the buttons, that was being handled over here, but nothing changed over here. We didn't do anything to change the state. Now, soon as we used to useState and recorded, then the Shadow DOM knew all about it and re-rendered. So, I hope that's given you some insight into how React re-renders and how the Shadow DOM works.

Video 17 – When Does UseEffect Run?

So, we've been talking about the hooks that React has. And we've seen that useState, if that's changed, then that causes the web component to be re-rendered. That means the function is going to be fired again. We're dealing here with functional web components. So, that means that

that is going to be called, that function is going to be called, that defines the web component. Now, useEffect can be used to control the life cycle of a web component. And let's understand when useEffect is called. So, useEffect takes two arguments. And if we have an empty array for the 2nd argument, the 1st argument is always a function.

The second argument, if it's an empty array, means that it will only be called after the first render. That useEffect won't be called again. If we miss out that empty array, then useEffect is going to be called every time there's a re-render. So, every time that functional component is fired, we're going to call useEffect. So here, we're watching props. And that means that if the properties of our web component have changed, then useEffect will be called. So, if we have useEffect watching the state every time the state is changed, for example, we call setState, then useEffect is going to be called.

And when the component is unmounted. We can also have cleanUp code, which is in the return value down here of useEffect. So, that cleanUp code will be called when our component is unmounted. So, useEffect can be used to control side effects like going out to fetch data from a remote URL. That doesn't change the Shadow DOM. So, there's no need necessarily for anything to be re-rendered. But by using useEffect, we can, for example, change the state and force a re-render. So, useEffect is very useful coupled with useState to control rendering and re-rendering of our web component.

Video 18 – Shopping Cart List With Complex Data

So, in this exercise, we're going to be dealing with data that's a little bit more complicated than last time. So, here we've got an array of objects and each object has a name, and whether it's got items instock. So, ' "apple" ' has got '2' items 'instock:' ' "pear" ' doesn't have any. What I want you to do is we're passing data in. So, we're going to pass in this data structure. We're going to set a minimum stock that we need to keep for it to be displayed. What I want you to do is to print out the list, but only with the items that have two or more instock. Okay? So, 'minstock' is set to '{2}' here. So, remove any item with less than that from appearing in the list. Okay, so good luck with that and you can see where we're going, that and this might be useful for a later project.

Video 19 – Shopping Cart With Multiple Lists

So, let's take a look at the starting point. And we've been familiar with rendering out this list. And now, what I want to do is when these buttons are clicked, I want those items to be put into a shopping cart, which will be rendered below here. Works like this. And when we've run out, clicking on it does nothing to the shopping cart. So, that's what I want you to produce. At the moment I'm giving you this where when we click, it gets decremented, but nothing gets put into the shopping cart. Let's take a look at the code. So, here's the code. The familiar bit, we're starting with the same data structure that you've already used. We're rendering change to the 'NavBar', and we're passing in 'stockitems'. We're passing this data structure in.

So, here's our 'NavBar', and we're getting that data structure. Now, there's some new pieces. We're going to use 'useState'. You've heard about 'useState', it keeps track of the state. There are two parts to it. The first part, let's focus on the 'stockitems' here because we're going to pass them in as the initial condition for this one. So, 'stock' will be initially, it'll be 'stockitems'. And then 'setStock' allows us to set its value. Now, when we set the value, it's going to cause a re-render of 'NavBar' because React knows that we've changed the state of something that is part of the DOM. And so, it's going to re-render. Okay, now we're using Buttons, we've seen that before. And here we're rendering out an 'updatedList', eventually here, at the end React.

We render out an updated list. So, we're calculating here. We're taking the 'stock'. So, that's an array. And we're using 'map' on it. And we're returning an element. And inside that, we're rendering out the '{item.name}:' the number of stock items. So, that's what you're seeing so far. Now, we want to, down here, render out a shopping cart, but I haven't put it in. So, underneath here, we need to create something like this. So, but with that, we'll render out the 'Shopping Cart'. Now, here we've got fragment because we always have to return from any component. We need to return something with a single parent. I'm putting a fragment in there. So, that's just empty angled brackets and the end as well.

So, now let's have a look what happens when we click on the button. We're going to call move cart. So, here is move cart. Now, 'e' will be whatever event is passed when you click on a button, okay? And I happen to know that 'e.target' will give us access to the 'innerHTML'. And what that is, is whatever is written on that button. So, for example, here, it will be 'pineapple: 2', that will be the 'innerHTML'. So, I'm using 'split' on it. It's a string so I can split it, and I'm splitting on the ' ":" '. And I know there are two; it's going to give me an array. It's going to put that 'split' string into an array with two parts. And the first part is going to be the 'name', and the second part 'num'. So, I can pick those up. Now, here what I'm doing is I'm going to modify the 'stock' state.

So, I'm going to take the stock, and I'm going to map it. So, the item will be of the full, one of these. It's going to pick out one of those. Okay, so that's what 'item' is, 'index' is just the index in that array, 0, 1, 2, etc. Now, I'm going to check if the 'item.name' matches the clicked-on button name. So, for example, the first one would be apple and date. If it was clicked, these would match. Now, I'm going to reduce the stock item, and then I'm passing item out. And so, that gets put into 'newStock'. So, this is a new array which has all the old elements in it. But with the one of them modified, the one that was clicked on. I'm putting the stock down by one just to show you how to decrease the stock.

Now, I'm going to set that stock state to '(newStock);'. So, I'm updating it, I'm resetting it to the new value. So, this is just demonstrating that. But now what I want you to do is in here, put logic in for moving that item into a cart and rendering the cart. So, you need to do some work here, there needs to be a cart, and I've given you that. You look up here, we've got in 'NavBar'. We've got a 'cart', and it's an empty array. So, you need to move items into this cart and then render them. Okay? So, it should look like, at the end, it should look like this. That we have an empty shopping cart at the beginning, but then we can put items into it. And we're decrementing and once they're '0', you can keep clicking it, it'll do nothing.