Introduction To JSX

Video Transcript

Video 1 – Introduction To JSX

JSX is at the core of the front-end framework that we have selected for our course. And as you will see, it is a curious thing. It allows you to write HTML-type tags elements within your JavaScript. Now that can lead to some pretty elegant solutions and some clean code when it comes to being able to create your components and be able to encapsulate your functionality in cleaner blocks as opposed to what you might have in HTML. Now because it's not truly HTML, there are some differences as well. And so, as we get started, we'll show you here our Hello World component, your prototypical component. We'll show you how to embed expressions into those tags that we will be writing. We'll highlight some of the differences, as mentioned, is very close, but is not exactly the same.

For example, when it comes to attaching a class to a component that you're writing in pure HTML, you would write class, and in this environment, you will write class name. And so, you'll see that there are some minor differences. However, your HTML knowledge, in general, will carry pretty well into this environment. We'll also take a look more broadly how to work with expressions within the JSX syntax. How to look at attributes and how do we work with attributes in this environment? And how do we create children? That is components that have, that are using other components that you have written. So, this is a good point here to get started with our JSX and look at some of the basic examples as we get going.

Video 2 – Introduction to Babel

JavaScript continues to evolve like all programming languages. And so, there's new advanced features of the language that are not yet part of the browser. And in order to use those, which at times can be pretty useful, we use transpilers. You can see here that Babel is one of those. A JavaScript compiler that transpiles from these new features onto Vanilla-type JavaScript. And so, here are a couple of examples. The very first one is one that makes use of something called the nullish coalescing operator. And you can see here what it looks like on the left-hand side. And you can see there the translation into something that the browser can understand. Here's another one for chaining.

I'm going to go ahead and paste that in, and you can see here that it has this funny '?'. And oftentimes, you would see that within or option to try to test to see if that had a value or was undefined. So, as you can see there, there are new features to the language that provides support for some functionality that the working groups lobbied for and that it made onto the official specification. And so, if we want to make use of this cutting-edge features or these newly released features. We often want to use this transpiler. Now, when it comes to React, React makes use of those extra features of the language.

And in order to avoid trying to, or trying to validate that all of the browsers and the versions of the browsers have had that added functionality. It relies on a transpiler. It does so for a number of things that are supported within the browser. Now, as you can see here we are in the editor and we're going to go ahead and write a sample application that sets up Babel. And you can use this as a template for the applications that you will be writing. So, I'm going to start off here by adding a couple of files. The very first one is simply going to be 'index.html'. And within this file, we'll add all of the libraries that are needed to support a React application.

This is a standalone application, and I'll say more to that in a minute. And then the second file that we're going to have is called 'index.js'. And here, we will add some React language. So, now let's go ahead and get started with our code. We're going to start off here with the HTML document, the header. So, you can see there the title for the page, and then a note simply to indicate that adding in within the browser and the way in which we're doing it is not high performance, and this is intended for learning. So, I'm going to go ahead and add the Babel library. As you can see there it is, and include a script that will be loaded when the page is loaded into the browser. Now, I'm going to go ahead and start on the body.

And the very first thing that we're going to write here is simply a heading tag to indicate that this a 'Template'. Then we will go ahead and note here the '< div >' element into which we will be adding our React code, and we will be adding our components. After that, we're going to go ahead and load the React libraries. As you can see there, we have both of them. This is, if you want to read a little bit more about this, you can read about the libraries and the documentation. And after that, we're going to go ahead and 'load our react component'. This is that extra file that we created, the ' "index.js" '. And as you can see here, we are deferring and making it of 'type="text/babble" '.

Once we do that, we're going to go ahead and close our '< / body >' tag, and then close our '< / html >' document. Now, let's go ahead and write our React code in our index.js. And as you can see here, we are going to write a pretty small bit of code. It's simply going to be 'ReactDOM.render'. Within it, we're going to go ahead and create a component. In this case, it is a very simple component. You can see there is just really a heading tag saying 'Hello World!'. And we're going to go ahead and add that to the '('root')' element, as we showed in the 'index.html'. This the element that we are targeting. And then we'll go ahead and close that line. I'm going to go ahead and save.

Now, as you can see here, I have the folder that we have within the editor opened on top of the browser here. And I'm going to go ahead and drag and drop my 'index.html'. And as you can see there, we get the 'Template' header. And if I open up my Developer Tools, you can see here that I have an error and that is complaining about an HTML or an 'XMLHttpRequest', then sort of cross origin type of challenge. You can read more about course. But ultimately, it is complaining about me loading files directly from the file system. Now, in order to avoid this, we need to run a small http-server. And this is what we're going to do next. Now, the server that I'm going to be using is an npm package.

And the name of that package, http-server, as you can see here, is a widely used server, small little server. If you're using another one that is fine as well. And I'm going to go ahead and run it with a couple of additional options. And the reason I'm going to be using those additional options is to make sure that the pages are not being cached. And so, that will be http-server '-c-0' And as you can see here, I've scrolled down on the documentation. And you can see there where it says, 'Note: Caching is on by default.' And that is, '-c-1' not '-c-0'. And every server will have some sort of capabilities to do this. And this simply helps us to know that the changes that we have made are in fact being served.

Now, the browser cache itself, but I will show that on the browser tools in a moment. So, now I'm going to go ahead and open up that new terminal. I'm going to go ahead and pull that window in. And I'm going to go ahead and type 'http-server'. I have already installed that server. I'm going to go ahead and enter here '-C-1'. Let me make sure and confirm the casing on that. That's actually lowercase. And as you can see there, my folder is being served at port '8080' on localhost. And so, I'm going to go ahead. Oh, and I pressed the wrong button there. So, I've opened up a new window, and I'm going to go ahead and type here 'localhost:' port '8080'. And as you can see there now, the page is being served.

I have 'Template', but also the React code, 'Hello World!' is showing up on the page, as you can see. So, as you can see there, we went ahead and wrote a basic page. You can think of these two files as a template where we added babel. And as mentioned previously, this is not for production later on when we see how to create React apps that bring in all of the tooling. This will not even be a consideration and will simply be added for you. In this case, we're doing it explicitly, and we're simply noting that this is not meant for production but for learning instead. We then went ahead and created our '< div >' that we were going to be targeting.

We added the React libraries. We then went ahead and identified where we were going to write, where we were going to grow, load our React component and that was going to be ' "index.js" '. We noted there to differ and use the 'type=' of ' "text/babel" '. And then within 'index.js', we simply created a new component. We added it to that element, the root element. And there you have a small little container that you can then expand, as you write your React application. You could write hundreds of components and bring them in in the same manner. And if you look at the templates that are used for the larger applications, they follow very much the same pattern. This is just a simpler version that you're looking at here. So, now it's your turn. Go ahead and enjoy, and make use of this template.

Video 3 – Adding A JSX Component on A Page

Let's write a barebones React application. We will do all of that in one single page with no additional tooling. So, let's go ahead and get started. I'm going to go ahead and open up the folder that I'm working on. I will go ahead and add a file. I will call it hello.html. And as you can see there we have an empty page. I'm now going to go ahead and add some 'html' boilerplate. And as you can see here, I'm going to go ahead and call the page 'Hello'. Now, after this, we're going to start adding our building blocks for our React application. The very first thing that we need is Babel, and we will use that because we will be writing some JSX and we need that to make the transpiling that takes place within the page.

Now, the next thing we need to add is we need to add the React libraries. And as you can see there, I have added both of them. The first one is called 'react.d'evelopment. The second one is called 'react-dom.development'. Both of these are served from 'unpkg', which is a service that is use widely in the billions per month of downloads, serve up MPM packages. Now, at this point, we have everything we need to start writing our application. So, I'm going to go ahead and add a '< script >' tag. This '< script >' tag is going to have some additional parameters. The very first one is 'defer'. That is, it's going to wait until the document is loaded by the browser. The second one is going to be the 'type='. And the type that I'm going to add here is ' "text/babel" '.

As mentioned, we will need that in order to be able to interpret the JSX that we will write. Now, inside of it, I'm going to go ahead and use a method call ReactDOM. And I'm going to use that to add an element onto the page. And as you can see here, I have the React documentation. And you can see there the render method that is part of ReactDOM. And it takes an element, a container, which will be a div on the page, and then an optional callback. So, I'm going to go ahead and add that now. So, I'm going to go ahead and enter here, 'ReactDOM', then '.render'. Then inside of it, I'm going to go ahead and add directly onto the parameter an element. And this element is going to be an '< h1 >', just the header.

And we're simply going to say 'Hello World!' in the famous tradition of first programs. Then we're going to go ahead and add an element. This is the element that is called container there in the documentation. And we're going to access that element using 'document.getElementById(' ')' And here, we're going to enter the id of the element that we have not added that element yet. And I do not need, by the way, the semicolon. We have not written that element yet. So, let's go ahead and write it. Simply going to be a '< div >'. And so, I'm going to go ahead and add that. I'm going to then give it an 'id'. And we will use that id to be able to target that element. And I'm going to call it ' "root" ' here.

Then I will enter '('root')' here. And so, at this point, we have everything we need for our simple barebones example. So, I'm going to go ahead now and drag and drop that file into the browser, as you can see there. And as you can see there we got HelloWorld! Now, let me go ahead and make an edit. So, you can see that it is dynamically updating. Now, let me go ahead and make some comments. As you can see there, we started off by adding Babel. You can see there the comments not to use this in production for small examples and learning this is perfectly fine.

However, once we get into larger applications and we start talking about production, we will do this using build tools.

Another point as well is that all we need when it comes to the React libraries are these two. Another one is that we targeted an element on the page and this will be common practice. This is our target that we use to introduce our content that we are producing on the script side onto the HTML page. And that this part here was used simply to build our component. So, as you can see, this is a great example to get a handle on the core building blocks in a small size, not a lot of code. Later, as mentioned, we will get a lot more tooling and more complexity in our applications. But ultimately, these are the core pieces. We have the library, we have a transpiler, we have script that is building components of the page and injecting them into the HTML document.

Video 4 – Embedding Expressions in A JSX Component

JSX can have embedded expressions just like strings in JavaScript. Let's go ahead and write an example to explore that capability. I'll start off here by creating a new document, and I'm going to call it expression or expressions.html. I'm then going to go ahead and add some HTML boilerplate. I'll call, or I'll title the document 'Expressions' as well. Next, I'm going to add Babel so we can do transpiling of the JSX on this page. Now, I'm going to go ahead and add a '< div >' onto the page. We will use this '< div >' to be targeted by React when we inject content onto the HTML document. Now, we'll give it an ' id="root" '. And I'll place a comment here. 'element used by react'. Next comes our library, React library, as you can see there.

Next, let's create our content. And I'm going to go ahead and enter here. This is where we will load our react component. And we will enter a '< script >' tag, and we will add to it two extra parameters. One will be 'defer' to indicate that we want it to load after the document has loaded. And the second one will be the 'type=', and the type we will enter is ' "text/babel" '. Now, inside of it, we're going to write the following. We will create a 'const name'. In this case, the name will be ' 'Peter Parker'; '. Then we will add our element. In this one, we'll be using JSX. So, we'll start off with an '< h1 >'. Then we will say ' Hello, '. And here is where we will add our embedded expression.

We do that with curly braces. And in this case, we're going to add the '{name}'. Now, all we need to do is to add that to the page, and we do that using 'ReactDOM.render'. Then we go ahead and enter two parameters. The first one is the 'element,', and the second one is the div that we are going to target or the element that we're going to target. So, we will enter 'document. getElementById', then the '('root');'. And as you can see there, I had entered a semicolon and the editor indicates that that's a mistake and in fact it is. So, now we've completed the page. We've added the needed packages which were Babel and React. We then added our target '< div >', which was ' "root" '. We then wrote, we defined a variable called 'name'.

Then we created our JSX, and embedded an expression in it, which was '{name}'. And then we added the 'element' to the page using 'ReactDOM.render'. And then the element that we're targeting in this case, is the one that is identified with the idea of ' "root" '. And we reference it by using 'document.getElementById'. So, let's go ahead and load that onto the page. And as you can see there as expected, we get Hello, Peter Parker. So, now it's your turn. Go ahead and write an element, copy this code. Then write in your script tag an element that has an embedded expression. You could play around with dates, you could play around with some arithmetic. But go ahead and have some fun with it and make sure you test there your understanding.

Video 5 – Adding Styles to A Component

Let's once again add an expression to the page. In this case, we will add a richer expression and we will also take into account attribute. Like for example, adding a class to an element. Let's start off by creating a new document. Let's call this one addStyle.html. And we'll start off once again with our boilerplate. We'll call this one style as well, or 'Adding Style'. We'll then add our transpiler. Next, our React libraries. Then the element we will target. Now, let's add our '< script >'. Then add the parameters of 'defer', and the 'type=' of ' "text/babel" '. Now, let's start off by creating an object. And that object is going to be 'user'. And the Properties inside the object are going to be 'firstName'.

And the 'firstName:' will be ' 'Peter', '. And then the 'lastName:' will be ' 'Parker' '. Now, let's go ahead and add another object. This one will hold the style. So, this will be 'const style'. And then the properties will be 'fontFamily:'. And the 'fontFamily:' is going to be very simple, just ' 'Arial', '. Then our 'color:' will be ' 'green' '. Now, let's build our 'element'. And we are going to enter first an '< h1', then a 'style='. And this 'style=' is going to be the object that we wrote for the '{style}'. And so, this is the embedded expression here. And now we're going to add the name. And we're going to go ahead and write here 'Hello,'.

And then we're going to enter the name. However, in this case, since we have an object, we can evaluate an expression. And this will be simply a function that we will write in a second that will format the name and we will pass in '(user)'. Let's go ahead and write that function. And the 'function' will be simply 'formatName'. And it will take an object which is '(user)', and then it'll simply 'return user.firstName. And it will concatenate that with a space. And then 'user.lastName'. Only thing left to do is to add the element to the document.

And we will add it with 'ReactDOM.render'. And then we will pass in the 'element;'. And we will get a handle on the div using 'document.getElementById'. And the id, in this case, is '('root')'. Let me go ahead and put in my line terminators, these two. And now let's go ahead and add that page into the browser. And as you can see there we get Hello, Peter Parker, in green. So now, it's your turn. Go ahead and write a page experiment by trying to build HTML dynamically and adding it into the page.

Video 6 – Using Expressions While Building Components

Before we saw how to embed expressions, Let's take a look at a broader use of expressions, not just embedded expressions in our JSX. So, let's start off by creating a new document. I will call it broader.html. Let's now add our HTML boilerplate, then our transpiler, then our target div plus libraries. And now, our '< script >' tag. Let me go ahead and close that. Now, let's get started with our code. And I'll start off by pasting into the page here some functions that we've written previously. You can see there that I have one to format the name and then I have the user object. Note that I've added one more property and this is 'age:', and in this case, '18'.

Now, say we wanted to write a function, determine whether a person was an adult or not. And in this case, we'll define adult to be anything over 18. Now, you could imagine we could write some strings and we could make that comparison. But could we do it within JSX itself, meaning be able to write that element and at the same time use expressions within the language? And as it turns out, we can. So, let's go ahead and write that function now. The 'function' I will call 'getGreeting'. The greeting being whether the person is an adult or not. And it will take an object, and in this case, the '(user)' object. And then we're going to make that comparison. 'if (user.age >=18)'.

Then we will start building that JSX and we will 'return' here in '< h1 >'. And we will say, for now, just say 'is an adult'. Then we can make it a little bit nicer by using the name, the formatted name, and we will enter here an expression. So, in addition to overall creating JSX using expressions, in this case, a conditional that we have, therefore, the comparison of the age. In addition to that, we're also embedding expressions within these strings. And in this case, we're going to format the name. And we're going to pass in '(user)'. And then we're going to have the else clause, which here I will simply use as 'return'.

You know, if you reach that point, it's the else condition. But in this case, I'm going to say 'is a minor'. So, at this point, we have our object, we have our 'formatName', and then we have the 'function getGreeting' that builds the element that we're going to return. And so, at this point, we can simply do that, 'ReactDOM.render'. And we will use 'getGreeting' here with '(user)' to build that element. And then we will enter 'document.getElementById' as usual. And then as usual we've called our element that we're targeting '('root')'. Then we will close that. And we have everything that we need to be able to evaluate that expression within the greetings function.

Which gives us a great deal of flexibility because we can use all of the parts of the language, not just within the embedded expression. So, let's go ahead and see what that looks like on the page. I'm going to go ahead and drag and drop broader.html. And as you can see there, Peter Parker is an adult. If we change Peter Parker's age and we reload the page, you can see there that now he is a minor. So, now it's your turn.

Go ahead and conceive of scenario where you have some conditionals, where you can take a couple of different branches. Perhaps you're doing a greeting based on the day, or perhaps based on the hour, you're doing, good morning or good afternoon or good night. And play a little bit with how you would construct this HTML dynamically, these elements, and putting them back into the page, and get a feel for how you can make those decisions. You can use the different parts of the language and you can work with elements that are being created dynamically.

Video 7 – HTML Attributes in Components

HTML attributes, a core component of HTML, have some twist when it comes to JSX. And let's take a look at an example to illustrate that point. As you can see here, we have a very simple document, starts off by adding the transpiler. Then it loads the React libraries, and then it defines the element that we will be targeting from our JSX in our code. Down here we have the placeholder for the code that we will be writing. And we will start off by defining a style. We will create an object to do that, and then we'll call the object 'style'. Then inside of it, I'm going to be adding font-family. And you can see here the specification and MDM for font-family.

Then font-size, which I have opened there as well. And so, let's go ahead and get started. I'll start off here with 'font-family:'. And the 'font-family:' is going to be ' 'Arial', '. Very simple. Then I'm going to give it a 'color:'. The 'color:' will be ' 'green', '. And then the last one is going to be 'font-size:'. And that has a dash. And the 'font-size:' is going to be ' '10 em' '. Now, as you can see there, we already have quite a bit of red underlining, meaning that the editor is telling us that something is wrong. And the reason for that you might have caught it already is that this type of naming violates property names within JavaScript. And in this case, it's one of the twists that I mentioned.

Since JSX is closer to JavaScript, that follows JavaScript conventions. Meaning that in this case, it uses camel casing for HTML attribute names. So, that means that this one here will become 'fontFamily:', and this one will become 'fontSize:'. But it's a little bit different than you would expect. So, if you're ever having an error with an attribute, it might be worth checking to see that JSX uses the same naming that you think it uses. In other cases, like you can see here for 'color:', there is no change. So, let's go ahead and write a simple element. And in this case, I'm going to go ahead and simply create a link element. And I'm going to go ahead and start that now. The link is going to be for MIT's web page. And so, I've entered there the address.

Then I'm going to give it the name that will be visible. And now I will add the style. And I will use an embedded property here to add the style. And inside of it, we will simply add our object. Now, the only thing left to do is to go ahead and add that element, the MIT element, the link to the document. And we will enter the reference to the '('root')'. And now after we add our line terminator, let's go ahead and drag and drop that document that's called attributes.html. I'm going to go ahead and drag that in. And as you can see there, we get the styling that we expected, even though we're using different, slightly different attribute names.

So, this is one to be careful of. Class name is another one that might surprise you. And that is because class is a reserved keyword within React. And so, this is one to test your understanding on. Go ahead and recreate the code that I have. Choose some attribute names that you think would conflict. And try and guess and see if you can come up with the name without looking it up in the documentation and make sure you put in that in your mind that there are some changes when it comes to attributes.

Video 8 – Adding Children to Elements

You can easily add children to JSX elements. And to illustrate that point, let's go ahead and construct an element that uses Bootstrap styles. I'm going to go ahead here and navigate to Bootstrap simply to show you where you can get the link for the style sheet. That's always on the front page. And later we will come back to the HTML that we will use to create a card. For now, let's go ahead and create a new page and we will call this page 'boot.html'. We will start out by adding our HTML boilerplate. And I will go ahead and add a '< title >', and I will call it 'Boot'. Then I will add my CSS styles. By adding this line, we get access to all of Bootstrap styles. Next, our transpiler.

Then I'll go ahead and add the '< div >'. This is the '< div >' that we will be targeting from our JSX. And finally, our React libraries. And now, let's go ahead and write our code for the Bootstrap element. We'll start by navigating to Bootstrap and copying the card syntax. You can see there what it's supposed to look like on top. And so, I'm going to go ahead and copy that. And I'm going to enter it in a variable called 'element'. I will enter it on the next line. Simply to be able to format it a little bit better. I'm going to go ahead and indent my HTML here and go ahead and fix this one so we can better see what the indenting is. And as you will note there, we have a hierarchy of elements.

And one of the nice things about JSX is that we don't have to think about how to represent those in JavaScript. We don't have to think about how to make the conversion back into the document, potentially if we had some parts of it that needed Javascript. All of that can be taken care of by the library. So, in this case, that's all we need to do, we can simply copy and paste the HTML from Bootstrap into, into this element here. And then after that, simply add it onto the page. And so, that's what I'm going to do next. And there I'm going to go ahead and add or specify the 'element' first. Then we would use 'document.getElementById'.

And the ID that we're targeting is '('root');', which is the one here, as you can see. And so, I've entered there a ' ; ' where I shouldn't. Let me go ahead and add it, one, where it should go. And so, now we have everything within our code. We have defined our element, we're using JSX. We have a hierarchy of children when it comes to the components. How those got represented internally? That is something we can defer to the package. And ultimately, we just take that element and we assign it to the page targeting the 'root' element. So, I'm going to go ahead and load that file onto the browser now, as you can see. And then, as expected, we can see our card.

Now, if I were to open up the developer tools, you'd see there a message asking me, if I meant to say ' 'className' ' instead of ' 'class' '. However, if you look at the card, the card looks just like it did in Bootstrap. So, that means that the library is being tolerant and forgiving me not doing className instead of class. However, if we were to use an earlier version of the library, and I'm going to go ahead and add those here. So, you'll see they'll show up on my left-hand side when I drag them in. And you can see there that I have 'react.js' and 'react-dom.js'. And I'm going to go ahead and modify my paths to point to those files.

I'm going to go ahead and remove first the URL part. And then I'm going to go ahead and remove the 'development'. Since the part that we have is just the file itself, not the development version. So, now I'm going to go ahead and reload the page. And as you can see there, we get a lot more errors. Now, the errors have to do or have to do with access to the file system. And we can address that by running a server. And I'm going to go ahead and do that now. 'http-server'. And now, I'm going to navigate to localhost port 8080. 'localhost:' port '8080', as you can see there. And if we select our 'boot', you can see there that now we can see the text, meaning that there's no issue accessing the file.

However, we still get the 'className' versus 'div'. And in this case, we get no formatting. So, an earlier version of the library has more issues when it comes to access to files when it comes to forgiveness on attributes on HTML elements. So, I'm not going to go back to the original version that we were running of the original library. I'm going to go ahead and undo. And now, if we go ahead and reload the page, we can once again see the element here. We can see the Bootstrap component, that's their column, Bootstrap. However, we have this message here. So, let's go ahead and get rid of that message by replacing 'className', 'class' with 'className'.

So, if I add that and then reload the page. Finally, we get the page without an error. So, this is one to take a look at closely, so you can identify some of the potential sources of errors. And also notice that as time goes by, the library can figure out more things on its own and also be more forgiving, and give you good messages back. Now, this is not always the case, but frequently it is so. So, try this, try to replicate some of the errors, try to play with some of the older libraries just to see the limitations that they have, and get comfortable with these side of errors, these type of errors because this is what you will need to do once you start to debug more deeply.

Video 9 – Forms - Capturing Data

Forms go back to the first days of the web, it is the primary mechanism with which we collect data from users. And because of it, it's important to understand that at its core. So, we will start with pure HTML in JavaScript and see how we can collect data, what are the events that are involved, and how that information is collected. We will then take a look at the same using the front-end framework that we're using, which is React. And we will look at it in two ways. One which is unmanaged, that is a lot closer to pure HTML. And then we will look at one that is managed, that takes advantage of the capabilities that the framework gives us, and so that we can save some of the work that we need to do otherwise. We'll then take a look in how to clean up our code.

Always a great thing when it comes to constructing our work and how to do that using some custom hooks within our forms. We'll take a look as well as how do we do validation, part of the normal cycle when a form is submitted and evaluating that content that has been entered and providing some feedback back to the user. Finally, we will take a look at a third-party plug-in, in this case, Formik, one of the most widely used plug-ins when it comes to forms that will do a lot of that work that we're now doing by hand, up until this point, and use it and do it using their framework. We will look at a basic form and validation using this third-party plug-in. So, with that in mind, go ahead and dive in. As mentioned, forms are a very important part of collecting information from users.

Video 10 – HTML Forms

Forms are an essential part of the web. It's how we collect data from users. So, to get started, let's create a simple HTML form with the basics. I'm going to go ahead and get started here by entering some HTML boilerplate. Then going to simplify my form and give it a title of 'Sign Up'. This will be a variant of your typical sign-up form. And then inside of the '< body >' we're going to add our elements. The first one is going to be a 'select' element. Inside of it, we're going to have a number of options. And I'm going to start off here with a simply, a simpler version of the tag. I'm going to copy it over four times since we will be entering four options. The very first one is going to be 'Freshman'.

The second one is going to be 'Sophomore'. The third one is going to be 'Junior'. And the last one is going to be 'Senior'. Next, I'm going to give the element, the 'id=' of ' "year" '. Now, let's go ahead and enter a few labels for our form fields. This is going to be the '< select >'. And after this, there's going to be some form field. I'm going to enter those in a '< div >' tag. And I'm going to go ahead and enter three of them. The very first label is going to be the 'Name'. The second one is going to be the 'Email'. The third one is going to be the 'Password'. Now, let's go ahead and enter our 'input' fields. And let's copy that one over three times as well.

And looks like I didn't get all of it. And now, let's give them some ids. The very first one is going to have an 'id="name" '. The next one is going to have an 'id="email" '. And the last one is going to have an 'id="password" '. Now, the last option we're going to add is going to be a check box to remember the user or give the user the option. And because I wanted it on a new line, I'm going to go ahead and enter it in a '< div >'. And here, I'm going to enter once again an input, but this time I will make it ' "checkbox" ' type. I will give it an 'id="checkbox" '. And I will enter the text, 'Remember me'. Now, before we go any further, let's go ahead and load the page in the browser, and let's see how we're doing so far.

And as you can see there we get what we expected, a pull-down. Then three fields and a checkbox. Next, let's go ahead and add a '< button >' so we can submit the form. We'll enter an element called '< button >'. Then 'onclick'. Let's see the 'onclick' event. We will enter a function called ' "handle()" ' that we will write shortly, And then give the text of 'Summit'. So, let's save that, reload the page. And as you can see there, we now get our Submit button. Now, let's go ahead and write that script that will contain that function. We'll start off here by entering our '< script >' tags. Then the function name, which matches the name we entered. And inside of it, we're simply going to write to the console the values that were posted by the form.

And so, we'll start off here with a 'console.log()'. And let me go ahead and minimize the sidebar here so we have some more room. And inside of it, we're going to write 'document.getElementById'. And inside of that, we're going to go ahead and enter the element that we're targeting. So, the very first one is going to be '('year')'. And we can access the value by entering a '.value);' there at the end. Now, let's do the same for the rest of the field. And let's next enter '('name')', then '('email')', then '('password')'. And then the last one is going to be the '('checkbox')'. Now, checkbox. Well, let me go ahead and enter first, '('checkbox')'. The access to whether that checkbox has been checked or not is 'checked'.

And so, that will give us a true or false depending on whether that was clicked or not. Okay, let's go ahead and save our document. Let's go ahead and reload the page. And let's choose here 'Sophomore'. Let's enter 'peter'. 'peter@mit.edu'. Let's enter the password here of 'secret'. 'Remember me', and let's go ahead and open up our console, so we can see our values. And I'm going to go ahead and make this a little bit bigger so you can see. And now, I'm going to go ahead and hit 'Submit'. And as expected there we get our form values back. 'Sophomore', 'peter', 'peter@mit.edu', 'secret', and 'true'.

If we were to remove that last option, we would get 'false'. Now, at this point, instead of simply writing to the console, you could go ahead and write to a database or validate your input. Many of the things that are done regularly. But for a first example, this is the basics, as you can see here, we created an HTML document. Then we added a number of input elements. We captured that data and then on the submit event, we are able to collect those values from the form. Now, it's your turn. Collect a form for some activity that you'd like to pick up data from users and walk through the steps that we just did.

Video 11 – HTML Form Tag

In our previous lesson, we passed the form data using a function and then using 'document.getElementById' to access the information in each of the fields. Now, we can also use a form element to wrap the elements and access the information that way. So, let's take a look at what that looks like. Going to go ahead and enter here an 'id='. And we're going to give it a handle, called ' "form" '. And we're going to wrap the rest of the elements inside of the '< form >' tag. Now, in addition to that, let's go ahead and pass the '(event)'. And we'll take it into the function as '(e)', which is the convention. And we can now dig into that event. And first list just show all of the data that was passed in as part of the event and we will enter '(e.target.form.elements);'.

Next list target each of those elements directly. We will enter 'elements.year.value);', just like we did before. And we will do the same for the rest of the elements on the page. The next one would be 'name.value);'. The next one, 'email.value);' 'password.value);' And just as before, the 'checkbox' is 'checked);'. That's how we access the value of that element. Now, let's go ahead and save our document, reload the page, and go ahead and Submit. And as you can see there, we get a flash. And the reason we get that flash is because by default the form posts, used to post to the server. And so, it reloads the page. But we can prevent that by simply entering 'event.preventDefault();', And that will stop it from reloading. Let's go ahead and reload the page, select. And as you can see there we get the rest of the data.

The first one is simply pointing to everything that we're passing on the event. And we will enter here peter, peter@mit.edu. Password of secret. And we will go ahead and post that. And I believe that should be lower in the page, and as you can see there it is. Let's just do it once more so it's visible. And you can see there we get all of the data as expected. This is the same mechanism that gives us the data. However, we're accessing those fields in a different way. When it comes to the code that you see online in many places, that postcode, you might see one of these two approaches. So, now it's your turn. Go ahead and rewrite the application that you've already written to collect user data, however, now access the fields using this different approach and where you are collecting and extracting using the form element.

Video 12 – React Form - Unmanaged

Let's create a simple React application. As you can see here, I have a simple application, really just a folder called SAMPLE, and inside of it a file called index.html. Now, inside of it, we're going to go ahead and add some boilerplate for a React application. And as you can see here, we have the usual things, the transpiler, the element we're targeting, the two libraries that we need. And then we're pointing to a file called ' "index.js" '. Let's go ahead and add that now, 'index.js'. And inside of it, we're going to start off by creating a component call 'SignUp()'. For now, all we're going to 'return' is an empty tag. And we're going to wrap that in a parentheses. Let me go ahead and create those tags now. And that's all we're going to do.

Then we will use 'ReactDOM' to add that element to the page. And we're going to use 'ReactDOM.render'. Then inside of it, we're going to use the element we just wrote, which is '< SignUp/>,'. And then we're going to target the element that we have in the HTML file using 'getElementById'. And then we are going to use the id which is '('root')'. Now, let's go ahead and add the same HTML we have been using for the 'SignUp' form. And as you can see there, we've done just that. Now, I'm going to go ahead and add a '< button >'. And this '< button', I'm going to have an 'OnClick>' event and notice the casing because we are now working inside of React and we use the camel casing as opposed to the usual all lowercase.

Then inside of that, we're going to have an expression. And this is going to point to a function that we will write called '{handle}'. So, let's go ahead and write that function, 'function handle()'. And we are going to go ahead and add the body and we will do a very simple thing. All we're going to do is write to the 'console'. And we are going to access one of these elements. And we will do one simply to illustrate that we can. And then we will use 'getElementById', two access that element on the page. And we want to, in this case, target '('year')', which is the very first one. And then we want to see the '.value);'. So, let's go ahead and save that page, reload our file. And let's go ahead and select.

And I did not put a title for my '< button >' or a string. I'm going to call it 'Submit' So, let me go ahead and reload once again. And now I'm going to select Sophmore, hit Submit. And as you can see there, we cease Sophmore as expected. So, this is very much the same approach that we used when we wrote a simple standalone HTML page. However, we now have it inside of a React application. So, this is an unmanaged, we're not using the state of React at all, an unmanaged application, and in the next one, we're going to see one where we start to use the mechanisms inside of React to be able to manage that state.

Video 13 – React Form - Managed

Let's add manage state to our form. That means that all of the elements that you see here on the page. So, the Year, Name, Email, Password, and the checkbox will now be tracked by the state on the component. So, let's go ahead and add those. So, I'm going to start off by adding a single one. And then I will leave it up to you to do the rest. I'm going to go ahead and start with the very first one. I will enter '[year,', then ' setYear]' as it's the convention for state variables, and then I'm going to go ahead and access 'React.useState();'. I'm going to initialize it to be an empty string. And I'm now going to go ahead and add it to the element. So, I'm going to go ahead and enter here 'value='. And I'm going to enter the variable that we just created, which is '{year}'.

Then I'm going to access or I'm going to define the 'onChange=' event. And inside of that definition, the expression that we're going to evaluate is a function that will set the value and that will be read from the event. And we are going to then use 'setYear()'. And we're going to access the value of that field. And we do that very much in the same way that we did when we were using pure HTML. We access the 'target.' of the event, then the 'value)'. And now, since we have state that's readily accessible when we handle the Submit button, we can simply write to the console directly the value that we have. So, we can enter here a message of '('year:', ' or a label of '('year:', ' And then I'm going to go ahead and directly access 'year);'.

So, let's go ahead and reload our page and select Sophmore. And let me go ahead and open up my Developer Tools. And I'm going to go ahead here and click Submit. And as you can see there, we can read the year as expected. Now, I'm going to leave the rest up to you, add state for the remaining elements. So, that's Name, Email, Password, and checkbox. Remember that checkbox access to the value is a little bit different. Meaning it's not just .value, but it's .checked. And so, you can review the HTML lesson if you're not sure of how to do that. But now it's your turn. Go ahead and try completing the form for the rest of the fields.

Video 14 – Custom Hook to Clean Up React Form

By the time you're done writing your form, there's a lot of repetitive work that has been done. And we can use a custom hook to clean some of that up. To clean up how state is defined, the sign and update it. How do we handle our onchange events? And we will be adding to each of those elements a name attribute that we will leverage when we write our custom hook in a moment, as you will see. So, as you can see here, I have added an additional file call 'useForm.js', and this will be the name of that custom hook. And let's get started with that now, I'm going to go ahead and call it 'function useForm()'. I'm going to take a set of 'initialValues'.

I'm then going to take those 'values' and add them to the local state. And I will use the function called 'setValues' to update that state. And so, this will use, 'React.useState()'. And we will take the '(initialValues);' as mentioned. We will then return an array with two values. The first one will be the 'values', that is the state. And the second one will be a function that we will use to update the state on the onchange event. And so, we will take 'e' as a parameter as usual for event. Then we will look or we will match the event 'type' of ' 'change' '. And if we match that, then we're going to set our values. And we're going to pass in an object and that will have the current 'values'. And the update will be the element that we're working on.

So, that will be '[e.target.name]:'. And the value will be 'e.target.value'. And so, as mentioned, we will use this to improve the way in which we define a sign and update those values that state and handle our onchange events. Let me take a look here. I have missed an 'i' in initial values. And now, let's go ahead and use, 'useForm.js'. And we will be able to remove the way in which we had defined our state before. And now, we will be defining and assigning values in the following way, '[values, handleChange]' is the function we will use. And this will use 'useForm();'. And into that, we will pass initial values for each of our elements. And the values will be, as you can see there, 'year', 'name', 'email', 'password', and 'checkbox' is false, initially.

And since we now have all of those values in one variable, we can enter here ' 'values:' ', and 'values' to be able to see what is posted when that form submits. Now, because we hold the values in values. We're going to have to enter a 'value.' for each of those elements, the value. And so, it's 'values.name', 'values.email', 'password', 'values.password', and 'values.checkbox'. Now, similarly, we can handle the 'onChange' event with 'handleChange'. And I will make the following changes throughout the form. And we need to make one last change because we're leveraging the target name, we need to add name attributes to each of those elements. So, I'm going to enter here, ' "year" '. Then on the next one, it's going to be ' "name" '.

So, the 'name' is going to be ' "name" '. Then the following one is going to be ' "email" '. The next one is going to be ' "password" '. And the last one is going to be ' "checkbox" '. So, now let's go ahead and save that form, reload the page. And it looks like we have an error. Let's go ahead and scroll to the top. And it says, 'useForm is not defined'. Then the reason is because we have not yet added it to the 'index.html'. So, let's go ahead and do that now. I'm going to go ahead and add an extra line here. And in this case, I'm going to enter ' "useForm" '. Now, let's go ahead and reload our page. And it looks like we have another error.

So, let's take a look at our 'useForm.js', and let's see if we can spot the problem. And yes, I can see that it's not useForm but userForm. Let's go ahead and reload. And it looks like we have our page back. So, let's go ahead and test it. I'm going to go ahead and select 'Sophomore'. I will enter here 'peter'. Then 'peter@mit.edu', password as 'secret'. Then select 'Remember me'. And I'll go ahead and 'Submit'. And as expected, let me go ahead and clear so we can see it better. We have our parameters except that if you note the 'checkbox:' is ' "false" '. Now, if I remove it, still says ' "false" '. So, that is not being done correctly. And if you remember the reason is because the 'value' is checked for that type of element. And so, I will leave this part to you to fix.

So, we have to overview. We created a new hook, custom hook that we use to clean up. How we created state? How we defined state within our form, how we assign it, how we updated it, how did we handle the onchange events? And we added an additional name attribute that we took advantage of in useForm. Now, this works pretty well. As you can see, the form is easier to read. There's less there being done. However, you have to. It is not meeting one of the elements in the way in which that value is accessed. So, this is now your turn. I will leave this up to you to fix and have fun digging in and learning about how you can access these values.

Video 15 – Validate Form

Let's validate our form. Validation can be quite involved, include third-party libraries. But in this case, let's take a look at the basics. As you can see here, I have simplified my form. In this case, we now only have Name, Email, and Password. And we will be looking at a basic example in a basic lifecycle when it comes to validation, we'll start off by creating a state variable of errors. And we will enter '[errors,' then 'setError]', which is the usual naming convention. Then we will use 'React.useState();' to hold those initial values. And we will initialize with the following '({nameError:' ', emailError:' ', passwordError:' '});' corresponding to those fields and initially set at empty. Next, let's go ahead and enter divs to hold those errors when they appear.

So, we're going to create a < div >. And inside of this < div >, we're going to add a style. We want that style to show up in a different color when it's triggered. And we're going to enter the '{color:' here of ' 'red'} '. So, that will give us a < div > with color of red. Then let's go ahead and enter our error message, in here we're going to go ahead and enter '{errors.' And the value of the corresponding, variable, which in this case is, '.nameError}'. And now, I'm going to go ahead and do the same for the other two elements. For email, I will enter the same, except that as you can see here, I've entered 'email' instead of 'nameError' and then the last one being the one for the 'passwordError', as you can see there.

Let's go ahead and save our page and reload on the browser to make sure that everything is okay, and it looks like it is. As you can see, we do not see any errors because they are empty at the moment. And so, now let's go ahead and work on our validation. I'm going to start off here by writing in a function that will 'validate( );', and this is a common thing that you would do before you handle the data. You want to make sure that the data is good data. And so, now let's go ahead and write that validation function. 'function validate( )'. Now, let's go ahead and start off with a very simple tasks. We're simply going to check and see if the '(values.name)' has been filled in.And if it has not, then we're going to set the error message.

So, in this case, it will be 'setError' and we will pass in '({...errors,'. Then we will add one more property to set the new value, which will be 'nameError:'. And then let's just say for now 'bad'})'. So now, let's go ahead and save our form. Let's reload the page and let's go ahead and Submit. And as you can see there, as expected, we get the validation. And let's go ahead and enter something. And now, that has not gone. So, one way we could address this would be to do an if-else condition. And if we are in the 'else' condition, then we set our error as empty. Let's go ahead and reload the page again. If we submit, we get bad as expected. Now, let's go ahead and fill in, and the error is gone.

Now, as you can see, this is a pretty simple example, as basic as it gets, you could do validation on a proper email address. You could do validation on the range of values. There's a lot more that you can do, and there's something else that is not apparent at the moment. And that is that when you start to reset several of the values, you have to be careful with how you reset your state. Or because of the asynchronous nature of state, you might only get the last value. So, I will leave it up to you to set the rest of the validation errors and to work out the best way to update the state when it comes to your errors to make sure you get the expected behavior. So, this is a good one to dig in and make sure that your understanding is fully what the behavior of the platform is.

Video 16 – Toolchains

Let's talk about toolchains. The very first thing to say is that you may not need a toolchain, as you will see in many of our examples, we're simply bringing in the directly, the libraries that are needed with a < script > tag. And this is certainly the easiest way to integrate React on to a website. And it's a great way to learn. That is, allows you to focus on the patterns, on the principles of React, and not on the tooling. However, you will hit a certain threshold upon when you will have to do a lot by hand. And when you're learning again, this is a great thing to do. But once you got it, you'd like to have additional support or you would have perhaps better arrangement, better organization of your files. You'll like to integrate packages from registries like npm, and in a more automated fashion.

You'd like to lint, you'd like to type-check. You'd like to have some hot reloading. That is, you can make your changes and without having to navigate to the site, the browser just automatically reloads that for you. Likewise, for packaging, bundling, testing, and development. So, as you can see, there are a multitude of things that you would have to do by hand if you were building it from scratch all the way up. And as your application grows, there are many more of these features that you'd want. As you can see here, some of those categories for the development server that hot reloading that we manage, that we mention package management, bundling, transpiring, testing, automation, and so on.

Now, there's some multitude of these toolchains, as you might imagine. And I'm listing three of those here, but you can navigate to 'reactjs.org' for more. Now, we will be taking a look here at Create React App because of its simplicity when it comes to creating a new application, it allows you to create a full-fledged application with most of the areas of support that we have discussed by simply entering one command, having one dependency, that is 'create-react-app'. You can then move into the directory that you've given the name for your app for, and then simply run 'npm start', and you're on your way. So, let's go ahead and use this application to create one here on our machine. As you can see here, I am in a terminal window on my desktop, and I'm going to go ahead and enter that one command.

So, 'npx create-react-app'. And then I'm going to call my application simply 'abel'. Now, when I do that. It's going to bring in all of that tooling that we don't need to concern ourselves with because all of that configuration, all of that arrangement is going to be done for us. Now, as you can see, the application has been installed. You might have seen some red there, by the way, and it's something we can ignore. It has to do with a different package manager. However, with npm, we can use this application without any issues. Now, as you can see there, it gives us some friendly instructions on how to get started. And I'm simply going to go ahead and move into my folder of the application that I just created.

And then I'm simply going to go ahead and enter 'npm start'. Now, as you can see, that application starts right away. And we see a template application that is up and running. You can see there that we get some instructions that say, 'Edit src/App.js and save to reload'. And that's what we're going to go ahead and do next. As you can see here, I have opened up an editor, and I'm going to go ahead and drag and drop my new application, the folder called 'abel'. And I'm going to go ahead and go over the directory structure. The one thing to note is that all of this has been created for us all of the configurations, all of the '.gitignore' that you can see there.

And we have a certain structure that is organized in the way that you see here. We have our 'node_modules', the long lists. We have our 'public' code already being displayed there. And then we have our 'src'. And per the instructions on the page, as mentioned previously, we can navigate here to 'App.js'. Then I'm simply going to replace the 'Edit' here with 'Abel was here'. When I save that, you'll see that the changes on this, on the browser page, are going to change automatically. So, I'm going to go ahead and save. And you can see there that 'Abel was here', updated. That is hot reloaded without us having to do anything.

Now, this is a simple example of the features that are built-in. You can see here that testing is automatically configured as well. There are many other parts to this platform for creating React applications. So, as you can see, that application that we just created had a great deal of what you see here in the diagram that we discussed before. If you want to learn more about it, you can navigate to Create React app. It is a widely used application. It is the most used toolchain that exists for this platform. So, this is a good one to take a look at and to learn some of its features.

Video 17 – Formik Form

Let's create a React application and call it sample, as you can see here. Now, that we've created the application, let's go ahead and move to the console and add 'formik'. Now, let's go ahead and start our application. We will do so by entering 'npm start'. And as you can see there, we get the default landing page. Now, let's go ahead and open up our default page. And we will go ahead here and clear out everything that's in it. We'll even remove the 'classNames'. Go ahead and save it. And as you can see now we have a blank page. Let's go ahead and remove the logos since we will not be needing it.

Now, we've added the formik package, but we need to 'import' it into our page here. We will 'import {useFormi} from 'formik' '. Now, that we've imported formik, and we have the basic building blocks here for a React application. I'm going to go ahead and bring in one of the forms that we've created previously. This is a very simple form that's simply sets a number of fields, the usual logging fields, which is name, password, email, and so on. And the reason I'm getting so many underlines is because JSX is a lot more strict than HTML. And we need to end each of those tags, and as you can see, we've gotten rid of the red underscores.

So, at this point, let's go ahead and save. And I'm going to go ahead and reload. And as you can see there we get, as expected, the fields on the page 'Name', 'Email', and 'Password'. Now, that we have the basics, let's start using formik. And we'll start off here by creating a component from formik. And we will use, 'useFormik( )'. This will take an object, and the components of this object will become apparent as we write the application. And we'll start off here with some 'initialValues:'. And each of those values correspond to each of the elements on the page. So, we have 'name:', 'email:', and 'password:'. And then in addition to that, we want to be able to handle the 'onSubmit' event.

And for that, we have a pre-defined function that formik is expecting and that is 'onSubmit:'. And we're going to take here 'values'. And the function that we are going to write is going to be a very simple one. This is simply going to write to the console. And in this case, we're going to write the form value. So, I'll write '('form:' ', and then 'values);'. Now, let's start working with the '< form >'. The very first thing that we're going to enter here is the 'onSubmit' event. And this one is going to map to the one we just defined within formik, and is going to be '{formik.handleSubmit}'. Now, in order to work with formik, each of the elements need to have a certain pattern.

We need to give it a 'name' property, an attribute, as you can see there. And then they need to have an 'onChange' event. And this 'onChange' is going to once again map to formik the handleChange. So, this is going to be '{formik.handleChange}'. And then the 'value', we'll go ahead and wrap here. The 'value=' is going to be '{formik.values.', and in this case, '.name}'. Now, the rest of the elements are going to follow the same pattern. And so, I'm going to go ahead and paste here the following. The 'name' will be ' "email" ', and 'email'. And then the last one, this is going to be 'Password', is going to be ' "password" ', and 'password'.

So, let's go ahead and save our form. Enter a few values, 'peter', 'peter@mit.edu', and then 'secret'. And let's go ahead and submit. And if we open up the developer tools, you can see there, let me go ahead and clear, submit again. You can see there that we get the values as expected from the 'form:'. Now, everything is being handled by formik, and we're making, or we're taking advantage of the predefined ways in which formik expects information. And wants this to define the onchange event and the values for each of those elements. So, now it's your turn. create a React app, add formik. Create a form of your own choosing, add formik as we have done, and catch that onsubmit event. Make sure that you can walk through the entire lifecycle of a form application now using formik.

Video 18 – Formik Validation

Let's validate a form using formik. We're going to be working on the same form we created the basics of formik form. And we're going to be adding one more function, the 'validate' function that we will use to validate the fields on the form. This is call 'validate:' app, and it takes in values which we've predefined. Then in the body, we are going to handle our validations. Now, this function is expected to return an object that holds all of the errors that were found. And so, we're going to first define that object. Then we're going to go ahead and take a look at each of the potential validations. In this case, we're going to do very simple validations. We're simply going to require the fields that are on the form.

And so, we're going to start off with '(!values.name)' and if that is not available. If that is missing, we're going to go ahead and add an 'error'. And the 'error' is going to be the error for 'name'. And in this case, we're going to put it a very simple message, which is that this field is ' 'Required'; '. Now, we will do the same for the other two fields. I'm going to go ahead and paste those in here. So, you can see there I'm checking for 'email', and then I'm checking for 'password'. Now, we 'return', as mentioned, all of those 'errors'. And then we let formik map those to the fields that we will define in a moment within the form where that message validation message can be passed into the user interface, into the browser page.

Now, let's go ahead and add the elements that we need and we're going to render, some elements conditionally. We're going to start off by entering '{formik.errors.name}'. And if that exists, we will add a '< div >' that will show the error. Next, let's go ahead and add a '< div >'. And we'll set the 'style' on the div. And inside of it, we're going to give it the '{color: 'red'}'. We want it to stand out. And we will then entered the error message and the error message will be '{formik.errors.name}'. Now, if the error doesn't exist, the third-part of the ternary. We will simply write 'null'. I'm going to go ahead and indent this more properly. And now, we need to do the same for the other two elements.

So, I'm going to do this first one for 'email', and then I'm going to do the last one for 'password'. And then the last one here. And now, we've entered a set of conditional elements that will display depending on whether there is an error message or not. So, let's go ahead and reload the page. And let's go ahead and enter here simply 'peter'. And as you can see there, it interactively activates us. We start to type. There is no error in 'peter'. However, there's an error for 'Email', and as soon as I start to type that in, it will go away. So, if I enter 'one', you can see there. And then again, now if I remove 'peter', the error is back. And so, you can see there that formik is handling the error messaging for us as long as we confirm, conform to the way in which formik wants us to define those attributes.

So, this is a very small introduction into the functionality of this widely used library, by far the most widely used library for forms. So, this is a good one to leverage when you're going to be doing extensive work on forms. Otherwise, you can take your own methodology and construct your own library that would give you perhaps a better fit into what you're trying to do. But here is now your turn. Go ahead and create a form, add validation to the forms that you've created before. And dig a little bit deeper into the functionality of formik, and see what additional things you might find valuable in the functionality of this library.

Optional Videos

JavaScript Tips and Tricks

So, as we've been programming in JavaScript, we've come across some interesting things that I certainly didn't know. And so, I thought it'd be fun to make some videos on, "Did you know this?" I particularly found I was making a mistake with filter. That filter returns an array and I kept forgetting that, that I thought it was just returning the object that I wanted. So, I put together just a few videos on "Did you know?" perhaps you can augment these as you go through the exercises yourself and as you go through the project. And maybe we can share those with the whole course.

ECMAScript 262 - How the Interpreter Runs

So, did you know that there was a standard for JavaScript called EcmaScript 262? And it still gives anyone instantiating JavaScript some flexibility in the way that JavaScript is processed. But the usual way is that JavaScript code will be Parsed. There'll be an abstract Syntax Tree formed, and this may reorder some of your statements. We'll see there's a thing called hoisting, where functions may be hoisted to the top of their scope. And that Syntax Tree is then run through an Interpreter to Byte code that can be executed. At the same time, there may be some optimization going on in an Optimizing Compiler that over time will improve the speed of that Byte code. Let's take a look at an example. So, here's a simple program that has a 'function main()' that returns ' "hello" '.

And there's a variable 'var x' that's given, instantiated to be '10;'. And then we call console and we fire the main function. Let's go to the first line of code that can be executed. So, it's this 'var x'. On the stack frame; the Global frame, we have main defined and main references this function on the stack frame that's on our left. The heap of objects is on our right. Now, we have x and it says it's undefined. That's because the 'var x' gets instantiated. But the value '10;' doesn't, the right-hand side, doesn't get instantiated yet. Now, when we execute that line 4, x does get instantiated. And now, we can call main(). And we'll see the main stack over here. And that returns "hello".

Notice we've got a variable here that's created by the interpreter that we don't specify a variable there, but there is a return value held for us. And as we exit the main function, remember anything, any variables defined inside the main function are going to disappear. So, we need this Return value to hold it for us and it sets main. Now, you will see to be hello, it returns hello. Now, that's interesting. But suppose now we try to execute the function before it's defined. Suppose we do this. Now, let's take a look at our Global frame. Now, let's go back to the first statement. We have x as undefined.

So, 'var x', the x there is recognized as a variable, but it's not initiated. Now, 'console.log (main());', let's see if we can execute it. Yup, it executes without problem. Here we've gone into main and we're going to ' return "hello" '. So, that's interesting what happens. Let's go back to the first what happens. Notice that main is already defined before we've even executed that first line of setting x equal to 10. It knows about this function main. This is called hoisting, and this is a function definition. And function definitions are hoisted to the top of their scope. So, what we mean by that is that main references, the function before we've actually executed a line of code.

Now, suppose we had x, x is just, it's not doing anything particularly, but suppose we put it at the bottom. Notice it's changed order with main, x is now after main on the Global stack frame. The function main is still promoted. So, we see that's returned. x is still undefined. We haven't reached that line of code. Now, we're at the line of code, and x gets instantiated to 10. If I write this function as a function expression. So, let's suppose I say 'var main = function'. And even if I put 'function main()', it doesn't make any difference. Let's go to the first line. You'll see main is undefined, x is undefined. So, it knows about main, the var main that has been promoted, it's at the top of its scope. But the actual function doesn't exist yet.

So now, we're going to try and execute the function, and we'll see that TypeError: main is not a function. So, at the moment it thinks that main is just some variable like a number and doesn't know what to do. So, it gives us a TypeError. So, function expressions are very different to function definitions, function expressions, the left-hand side gets hoisted, but the right-hand side does not. And it's actually the same here. You'll see that x has been hoisted, the program knows about it, but it hasn't been defined yet. It hasn't been assigned. Okay. So, that's "Did you know this about JavaScript?"

How to Filter Lists

Did you know the most common error when using the filter callback on an array? It's a very useful function. Let's suppose we want to pick out this number 3. So, this 'result', we'll take 'A.', and will run 'filter' on it. It takes an '(item', and we'll use '=>'. And we want to pick out the 'item' that's equal to '3'. So, the first thing is remember, map, you need to actually return something here. You just need to have true or false. As you list through the items, you compare the item with something or you compare its index possibly. And it just, if it's true, then that will be picked out. If it's false, it will be filtered out.

So, that's our filter and we've got our result. And so, now we can use the number. We've picked out the number 3, and we could use it somewhere. Or could we? Look. It's printed out, and it's an array. 3 is, it returns an array, and if there's only one element in it, then it's still an array. And if you need to use it, you need to get at that first element. So, mentally, I'm often thinking I'm just picking out one element. Now, I've got it. But I don't. I need to do that. I need to get up the first element to get its value. Okay. I hope you don't make the same mistake, but whenever I use filter now, that's what I think. Be careful. It's returning an array.

How to Use Reduce

So, did you know how to use reduce? You've heard about MapReduce, but not many people remember how to use it. Here's how I remember. So, I want to add up these numbers in this array. So, I'm going to write the reducer function. So, I'll 'let reducer ='. And I'm going to use fat arrow notation. But I'm going to have the '(accum', and the 'current)' value. And what I'm going to return is 'accum+current;'. Okay. So, that's a reducer. And now we can write. So, the 'result' is we're going to call 'A.reduce'. And we feed in the '(reducer,' function and the beginning value. So, let's suppose I want to add these numbers to the number, say '3)'.

We're going to start at 3 and add 1. That's 4, 2, that's 6, 3, that's 9, and 4 is 13. The result is 13. So, this works. Let's just put '0' in there. That's more normal. And we see the result is 10, which is correct as well. So, that's the way I remember it. I break it into two. You can obviously put it on one line that we could take this and we could put it in here. So, reduce and then instead of reducer, so we put the function. So, this is the function. Now, we want to end that function. Let me, I'm going to put curly braces in here just so you can see the end of that function.

So, there's the reduce, but it takes a second argument. And there it is. But I prefer it as we had it before. I prefer that. It's clearer to me. Now, remember, you can have arrays in here. You can start with an array, for example, that you're doing something with. Or you could have this initialized, for example, with an object that you're adding things into. In this case, just we're adding numbers. So, we start with the number '0'. Okay, but that's reduced. I hope I've taken some of the fear out of it. That it's a fairly straightforward function, straightforward callback, and very useful.