Week 18 - Bad Bank Exercise

Video Transcript

Video 1 – Introduction To Bad Bank Exercise

Individual lessons give you an opportunity to master a technique. Now, bigger applications expose you to the breadth of considerations when designing an application. And so, we wanted to give you some of those as well, although simplified for the purposes of the course. In this case, we wanted to show you a bad bank example called Bad Bank because it has no security. At the start, we'll take a look at some of what we will use, and we will do that in a standalone fashion. We will take a look at routing, at contexts. How do we provide a memory context for all of our screens? And then how we will style, we'll then move into the actual application design and construction. And we will take a look at all of the components as we build them.

So, a navigation bar, how do we do routing? How do we set the context? How do we style our application? And then ultimately, we will look at some of the screens. We will build some of them. Build Create Account, and then give you an opportunity to write out the remaining screens. Now ultimately, once you are done with the basics, with the rough design of the application, we will give you an opportunity and a challenge to be able to redesign this application and create something that has better design principles, better encapsulations, better modularity. And so, here's where you get to learn even more on your own and to explore some of those decisions that turn out to be key over time.

Video 2 – Routing Basics

In the early days of the web, when we were navigating through a site in our browser, and we wanted to request a subsequent page, we would click on a link, and that link would trigger a request by the browser to the server somewhere out in the world. And you would see that page be refreshed. That is, if you were navigating to a site that was slow, you would see your page temporarily perhaps go blank, and then that new page slowly load into the browser. Now, this worked very well. However, it wasn't a very good customer experience, especially when compared to a native desktop application. In time new technology came to the browsers. The model, in the pattern of Ajax, Asynchronous JavaScript and XML, and others HTTP request and more recent ones that have given the browser the capability to make that call in the background.

That is, your UI remains as is the browser itself can reach out and request that data, and in the background update and integrate that information. Now, you can call out for data, but you can call also for anything else you like. And so, this is the model that has been called the single-page application that is very similar to what we used in the past as native desktop applications. And so, that is where we're going to start. We're going to start by setting up some routing. We're going to

have the following architecture within our site. This is our information architecture or design. We're going to have an index.html where we will reference the following files. Index in three components.

Index will define the routing, and then we will have three components that we will load through a navigation bar that we will create. And those components will be home, about, and products. So, let's go ahead and get started. As you can see here, I have a browser on my right-hand side, and on the left, I have the editor. And as you can see there I have pretty much HTML boilerplate code, a header, a body, and a few placeholders. I'm going to go ahead here and add my transpiler. Then I'm going to go ahead and add the element that I'm going to target. As usual, I will call it ' "root" '. Then our libraries for react and the router, as you can see there. And finally, the pages that we mentioned, we were going to be loading. I have ' "home.js" ', I have ' "about.js" ', I have ' "products.js" ', and I have the ' "index" '. Next, let's go ahead and add those files.

We're going to start off here with 'index.js'. Then I'm going to go ahead and add 'home.js'. Next is going to be 'about.js'. And finally, 'products.js'. So, now then we have all of our files, empty files. I'm simply going to test serving up this page to confirm that I can, in fact, load this. Okay, and you probably noticed that I have this file here that I did not name properly. So, I'm going to go ahead and enter that now, 'products.js'. And now, we have all of our files. So, I'm going to go ahead and start up my server. Now, I'm going to go ahead and navigate through the browser. I'm going to go ahead and enter here 'localhost:8080'. And as you can see there, we get 'Routing', which is the title that we gave the page here, on line '5'. But we get nothing else because our page is empty.

We have, we're pointing to files, but those files have nothing. Now, let's go ahead and work on the 'index.js', which as mentioned, will be the place where we define the routing for our page. And I'm going to go ahead and start off by creating a component called 'Spa()' for single-page application. Inside of it, we're going to reference 'Route', 'Link', and 'HashRouter' from the routing library. As you can see there, we access that through 'ReactRouterDOM.Route;'. And we do the same for '.Link;' and '.HashRouter;'. Next, let's add our 'return' tags. I'm going to go ahead and start here by creating the parentheses within which I will enclose it. And now, I'm going to go ahead and add my '< HashRouter >'. After that, I will add a '< div >' to hold my tags within.

Then the next thing that I'm going to do is simply a header. I'm just going to put a header saying 'Hello World', so I can see that I've landed in the right place and that I am looking at the content of this component. After that, I'll start to define my links. As you can see here, I have the core, the base, the root, path pointing to 'Home'. The next one is going to be 'About' in, through this mechanism, we are defining our paths. The next one is going to be 'Products'. Then after that, I'm going to put in a horizontal rule simply to separate our content that we have at the top of the page versus that where we, through which we will load the components, which will be in the next part. And here, I will define my routes.

My first one will map, the route one will map to the '{Home}' component. The second one will be 'about', and that will map to the '{About}' component. And likewise, for the third one, 'products' to the '{Products}' component, I'm now going to go ahead and start closing my tags here. First my '< /div >', then my '< /HashRouter >'. And that defines our routing mechanism as well as the content that we're going to have on the top of the page. So, we're going to have here a header.

Then we're going to have a navigation bar that we are separating with these two little dashes here. And the reason we're doing that is because we're not using styles yet. And if we don't do that, they're all going to appear right next to each other, so it'll be hard to tell them apart.

And then the bottom part is simply where we set our routes. Next, let's go ahead and create our components. We'll start by creating our 'Home()' component. And to this, we're simply going to 'return' the, a small message here denoting where we are. And so, I'm going to enter an '< h3 >' header here. And I will say 'Home Component' here. Component. And we're going to do pretty much the same thing for the other two. And so, I'm going to go ahead and copy this over. And I'm going to go ahead and change this for 'About'. And then we're going to go ahead and do the same for the last one. And this one will be 'Products'. And this will be our message coming back. So, I'm going to go ahead and save it.

And I will go ahead and reload my page and see if I got my syntax right, and if we can see the site. I'm going to go ahead and do that now. And I do not see anything. Let me go ahead and take a look at my console. Ah! I can see here that I did not save my file. Let me go ahead and reload it once again. And I'm still getting an error. So, it looks like when I created that file, I created a folder as well. Let me go ahead and try that. And now, we have no errors, but we still cannot see anything on the page. And let's go back to index and see if we're missing anything. And yes, now see it. I am missing the loading of the 'Spa' component which loads the rest. So, I'm going to go ahead here and enter 'ReactDOM.render()'. Then we're going to enter the '< Spa />' component 'Spa'. And I'm going to go ahead and close it there.

And then we're going to reference the element on the HTML page that we're targeting, which in this case, it's the '('root')' element. And I'm going to go ahead and save that file, go ahead and reload. And as you can see there now we have the application we expected. We have routing across the top which matches our header. Then we have our navigation, which is 'Home', 'About', and 'Products'. And by default, we have loaded here our Home Component. Then if we click on 'About', you can see there that we see About Component. And then if we click on 'Products', we see the same. We can also navigate using backwards and forwards in the browser itself.

And so, the mechanisms of routing, although they are local and being managed by the page, are deeply connected to the browser. And it creates to write history. And we can navigate through as if we were loading pages in the old days. We're now handling everything within the browser. Do handling that, loading and unloading. So, now it's your turn. As it turns out, routing is one of the core components when it comes to building sites. So, this is one that you want to practice, replicate the exercise that you just saw. Play around with building the navigation, loading the components, and make sure you understand the basics of routing.

Video 3 – Context Basics

In our previous lesson, we created a SPA, a single-page application. That is a page that after your initial request to the server, no longer loads in its entirety, all of the subsequent changes and calls

to the server are handled in the background. We used routing to be able to handle the clicks on the navigation menu, which in this case was Home, About, Products to be able to load the corresponding components onto the page. Now, in this case, we're going to extend the functionality of that application by adding a shared context. This is a shared context that all of the components are going to share with index.js.

So, let's go ahead and move to our editor and do that. As you can see here, I have the same code as before. I have an index.html through which all of the subsequent .js files are loaded. And then I have an index, a 'Home' and 'About' 'Products'. These here, the three of them Home, About, and Products are components. And then we have our index page through which we define a routing and load it and mapped our paths to our components. So now, we're going to go ahead and add a user context. I'm going to start by creating a variable called 'UserContext'. And then we are going to create that context through 'React.createContext', and we're going to initialize it '(null);'. Next, we're going to add a tag for that 'UserContext'.

And in this case, we're going to use the 'Provider'. And we're going to initialize a 'value' here of a simple object that is going to hold some information that we will then access through our components. And that object is going to hold. Let me do the object curly braces within the expression we're evaluating. And so, our 'users:' are going to be what we are going to share. And this is simply a object that I chose out random. And so, this list here of the array is going to have a number of users, but I will initialize simply with a user called, '['peter']'. And then we're going to go ahead and close that tag. And I'm going to go ahead and wrap our routes with it. I'm going to go ahead and indent it just so it's more apparent.

And so, to review, I have created the 'UserContext'. As you can see there, using 'React.createContext'. Then I added this to my tags. Using 'UserContext.Provider' added an initial value and wrapped all of those routes with it. Now, I'm going to go ahead and move now to home where we will access that context. I'm going to start off here by creating a variable called 'ctx' for context. Then we're going to '.useContext'. And I'm going to reference the '(UserContext);' we created. And now that we have a handle on it, I'm going to go ahead and stringify that object using 'JSON.stringify'. And inside of it, I'm going to reference that object that we added, which was '(ctx.)', or in this case, the handle is '(ctx.' and then inside of it, I'm going to reference '.users)'.

So, let me go ahead and save this. And I'm going to go ahead and reload the page. And as you probably noticed here on the navigation menu, on the path, I am on Products at the moment, then I need to be on Home. And as you can see, as expected, once we load the home component, we can see [''peter'']. And once again, that comes from that shared context. And what we are referencing is that value that we're sharing with all the components. Next, let's go ahead and add some content or add some users to our context. And so, we're going to go ahead and get started here. I'm going to go ahead and copy this line. And I'm going to go ahead and do this work within products. So, every time we load the product's component, I'm going to add a user to our context.

And so, the very first thing that I'm going to do is I'm going to get a handle on the context using the same thing we did before, which was just 'React.useContext' and then the '(UserContext);'. And now we're going to go ahead and add into that context. So, we're going to access the context,

then users, then we're going to push into it. And here is where we're going to add a random string. And here is some code that I wrote previously which simply generates a random string that we will use. Now, once we've added that random string, then we're going to once again add an expression here that is just going to be 'JSON.stringify. And inside of it, we're going to access 'users'. And now, I'm going to go ahead and save that page. I'm going to reload the page. And when I go to Products, you can see there that I have an additional string. Now, every time I load that component, that string is going to be added to 'users'.

So, if I click on it again and again and again, you can see that those users are being entered into that array. Now, if we go to About, in About, we're not looking at the context, so none of that information is there. But if we go back to Home, you can see the list of users once more. Now, I can click on Home all I want. But because I'm not adding to the context, nothing changes. Going back to Products. You can see there that I've added yet one more. So, as you can see, creating a context that is shared amongst the components can be pretty useful to have that shared state. This is something that now it's your turn to practice. Go ahead and replicate the code, come up with additional scenarios in where you could use this shared context. Perhaps one of them might be where you are keeping counters between different tabs. And you have an additional one where you show the counters for all of the tabs. So, now it's your turn. Go ahead and dig into the code and have some fun.

Video 4 – Style Basics

In our previous lessons, we created a single-page application. We added routing, then we added a shared context for the components. And now, we're going to go ahead and add styles. And to illustrate how we might do that. We're going to go ahead and add bootstrap styles, which will be a reference to a script in index.html. And then we will replace the navigation that we have added to index.js with some syntax that will style the navigation links. And we will add that in a component. And we will add that within a file called nav.js. So, let's go ahead and move to the editor. As before, we have our files which are 'index.html', 'index.js', 'home.js', 'about.js', and 'products.js'. In here, I'm going to go ahead and add the reference to Bootstrap styles.

I'm going to go ahead and wrap the content so you can see it there. You can see there that is simply a link to a style sheet. I'm then going to add one more file here. And I will call it ' "nav.js" '. And this is where we will create our navigation content. I'm going to go ahead and line these up. I'm now going to go ahead and create that file, 'nav.js'. And as you can see there, it's empty. Now, to get the syntax that we need to be able to style that navigation, I'm going to go ahead and go to bootstrap.com. And I'm going to go ahead and show you how you can find that if you go through the documentation, in this case, we're interested in a nav.

And if you scroll down, you will find your Navs here. There's also navigation bars, but we're going to use something simpler for this example. And as you can see here, it's the type of syntax that you can enter. I'm going to start out by writing the 'function' for my component. And I'm going to call it 'Nav()'. Inside of it, I'm going to enter the tags. Now, I'm going to go ahead and enter my

tags. I'm going to start off here with the unordered list. And notice that I'm using the same syntax that I can see here within Bootstrap. And above is a preview of what it will look like. I'm then going to go ahead and add an item. And the item that I'm going to add first is the root, the link to 'Home'. The second one that I will use is 'About', and you can see there the path for it as well.

And notice that I'm using the same classes as Bootstrap. So, that is instead of using the default, I am using the classes that Bootstrap is denoting here. And I am changing 'className' because that is what is required within this environment. Class is a reserved keyword, and so we use 'className'. I'm then going to go ahead and continue here. The next one is 'Products'. Then I'm going to go ahead and close my unordered list. And the last thing that we're going to do is we are going to indent here our content so we can see the hierarchy of tags. Now, let's go ahead and save our file and go back to index. And in this case, we can replace all of this with our new component, which is '< Nav/>' and so, I'm going to go ahead and enter that.

Close my tag. And I'm going to go ahead and reload the page. And we're not seeing anything. So, let me go ahead and open up my Developer Tools and read the message. Oh, and I can see that link is not defined. And that must be because I am defining link inside of 'Spa' as opposed to outside with global scope. Now, I could change my 'nav' to 'url's instead of 'link', and that would work. However, in this case, since we're simply illustrating styles. I'm going to go ahead and move this to the outside simply to demonstrate a quick fix. And I'm going to go ahead and reload. And as you can see now, we get very different styles.

The font has changed. The navigation bar is much more space, and we can link just as before. And we have functionality that we expect, and the styling that we expect. So, go ahead and try it now for yourself. Go ahead and add the styling, make sure you can get some styles. You can also change the navigation bar from links to URLs, like you. So, here on Bootstrap, so that is the 'a' at 'href', and then closing tag. And you'll see that you get the same behavior. So, now it's your turn, go ahead and have some fun creating some styles.

Video 5 – Bad Bank Exercise Overview

In the following series of videos, we will be creating an application called Bad Bank, so-called because it's a bank with no security. However, it will give us the chance to create a full application. We will have navigation bars, styles, components that we will load on and off, and a container. It will give us a chance to practice a routing, decisions on context, and what styles we use. How do we design our components, which are the parents, which are the children? Decisions on where to keep state, which is a difficult one when you're getting started, your forms, and how do you handle those events. We will be initially creating a path using routing where we will decide on which components to load. And more broadly, the architecture information of our application, the information architecture will be as follows.

We will have a number of components across the bottom of the page, as you can see there. That will be integrated through an index.js page. And we will then have a global context to be shared by all. We will also then have a navigation bar, as you can see here in the diagram. As mentioned,

we will be providing routing, which will give us the capability to load and unload all of those components. We will then be creating a shared context which is that state that is shared by all. And finally, we will style our application using Bootstrap so that we can be realistic about the type of decisions and content that we have to consider when we are creating an application and deciding on styles. So, with that in mind, let's go ahead and get started.

Video 6 – Create Bad Bank Application Files

One of the things we're always fighting in the world of software is that of complexity. If it gets too high, we struggle to try to understand what's taking place. Bad Bank has enough in it that we will build one piece at a time, verify its functionality, and then move forward. And in general, this is a good practice. So, with that in mind, I'm going to go ahead and follow the information architecture that we have overviewed. And I will simply create the files that are highlighted. I will leave the rest to you. You can see here in my index.html is the place where I will define the files to be loaded. And now let's go ahead and move to the editor and start that process. As you can see here, I have increased the size of the window of the editor because in this case, we're simply going to be loading the files, and showing the header within the browser is going to be enough.

And at this point, I have a single file in my application. If we take a look at the files, we have a single one which is index.html. And in this case, I'm going to go ahead here and build up the boilerplate. And as you can see there, all we have is what you would expect, the header, the body, and a title that says 'Badbank'. Now, I'm going to go ahead and add the transpiler, as usual, then the element that we're going to be targeting, which is ' "root" ', then the libraries, in this case, we will be needing the libraries for the router. And finally, the files that we're going to be using. As mentioned, I will not create all of them. But at this point, I will go ahead and move to the console. as it will be faster to do it through the command line than to do it through the tool in the editor. And so, let's go ahead and get started. I will be using 'touch', and in this case, the very first one will be 'context.js'.

Now, if we go back to the editor and we expand, we can see here that we have all of the files that we have just created. All of them are empty files since we simply created them using touch. So, I'm going to go ahead and remove the files that I will not be creating, but that will be left for you. So, at this point, we have everything we need. All we're doing is creating the file. So, I'm going to go ahead and fire up my webserver to be able to load that application onto my browser. And I'm running that at port 8080. So, I'm going to enter here localhost:8080. And as you can see there, we get the Badbank in the tab title and nothing else because we have not done anything else. So, now it's your turn. Go ahead and replicate the code that I have added. Go ahead and add the additional files. And then we will move on to our next step.

Video 7 – Bad Bank Navigation Bar

Next, we're going to add the navigation bar. As you can see here, that will be the navigation bar across the top of the page. It will have pointers to the components that we will eventually write, and the components will be Create Account, Login, Deposit, Withdraw, Balance, and All Data as you can see there. And for now, we're not going to write any styles. We simply want to confirm the functionality of writing a navigation component that we can see within the browser. So, let's go ahead and move to the editor. And here we'll start off with the navigation component, which is navbar. As you remember, our files are empty for now. So, I'm going to go ahead and start writing that component. I'm going to call it 'NavBar()'.

And then I'm going to go ahead and start writing my 'return' tags. And I'm going to wrap the paths that I'm going to write in a moment in a fragment. We do that by entering an empty tag. Now, I'm going to go ahead and enter those links. And as you can see there, the first one is the root path that is pointing to 'Badbank'. So, that's the landing page. The next one will be for 'Create Account', the next one for 'Login', the next one for 'Deposit', the next one for 'Withdraw', 'Balance', and 'AllData. Next, I'm going to write the parent component, which will eventually be the parent to all components. And I'm going to enter that in index.js. So, I'm going to go ahead and expand my files here and then start authoring index. I'm going to go ahead and call that component 'Spa()' for single page application.

And inside of it, we're going to write our jsx. And inside of it we're once again going to enter a fragment that's an empty tag. Then I'm going to add a header. This is simply so we can have something to look at when the page loads. And that's going to be an 'h1'. And I'm simply going to say 'Welcome to bad bank'. And below it, we're going to add our navigation bar. And so, we call that component 'NavBar'. And so, I'm going to go ahead and enter that now. And so, now that we have the 'Spa()', the single-page component, all we need to do is add that component to the page and we do that by adding it to the DOM. And so, we start out with 'ReactDOM.render'. Then we include our new tag, which is the '< Spa/>'.

And then, we target the element in the HTML page that is going to hold our '< Spa/>'. And we do that by entering 'document.getElementById', and the id of that element is '('root')'. So, now let's go ahead and save the page, fire up our HTTP server. And once we've done that, now we're going to go ahead and hit the '('root')'. And as expected, we get there the header, which is Welcome to bad bank. Then we get that navigation bar, which right now has no styles. So, it doesn't look that good. But you can see there that if we mouse over, you can see that it's targeting each of those components that we will write. So, Create Account, Login, Deposit, and so on. So, now it's your turn. Go ahead and add your navigation bar.

Video 8 – Bad Bank Routing

Next, we're going to add routing to our application. We'll start off by adding our routing components to context.js. Those components will be leveraged within index, which will define the routing paths and the mapping to the components as you can see here in this illustration. We'll start off at a high level by simply defining those router components which come from the routing library. So, let's go ahead and move to the editor and do that now. We'll start out with context.js as mentioned and we're going to go ahead and copy and paste those three lines of code. These components are coming from the react router library that we loaded in our first page on our HTML, at the beginning of the project. And from here on out, we will have then access to router link and hash router the components from the library.

Next, we'll move to index.js and here, we're going to go ahead and make use of those components. We'll start off here by entering the hash router tag, go ahead, and start a tag and then, enter hash router and I'm now going to remove the empty tags since we will be wrapping the rest for the hash router. I'm also going to remove the header since we want that navigation bar to be at the top of the page. So, I'm going to go ahead and wrap here and then, I'm going to go ahead and enter my first route and go ahead and remove the sidebar to give myself more room. The first route will be to the root path and so, I'm simply going to enter the root path, and this is going to be the default, and it's going to point to, the home component, and I'll do the same for the rest of the components on the navigation bar. So, create account login, deposit, withdraw balance and all data. Now as before, I will leave the components from login to balance to you and I will simply create here placeholders for create account and all data. At this point, we're simply adding placeholders since what we want to verify at this point is the routing.

Now that we've created the routing, you can see here we are pointing to components. So, we're pointing to the home component, to the create account component into the old data component. So, the next thing that we need to do is we need to go ahead and create those components. So, I'm going to go ahead and expand my sidebar and I'm going to go ahead and get started here with home and I will create my component function home, return, and inside of it, the tag that we're going to write is simply an H1 tag and all we're going to say here is home. We'll do the same for the other two. And so, the next one that I'm going to do is create account and I will simply replace here create account. Do the same for this part and the next one the last one is all data.

Before we load the application onto the browser, let's review what we did. We went ahead and we added the components from the library into context.js. Then, we used those components to create a routing mechanism with an index.js that pointed to the following components: create account, all data, and home.js. And so, now let's go ahead and test this within the browser. Now, let's go ahead and reload the page. And we do not see any content. So, let's go ahead and take a look at our developer tools. Let's scroll up to the first error and it says home is not defined. That appears to be lowercase so looks like I entered a lowercase component here. I'm going to go ahead and go ahead and wrap as well so you can see all of it. Now let me go ahead and save that, reload the page and as expected we can see the navigation bar across the top and our first component to home component being loaded.

Now I also added create account, you can see there that we now get create account and then, all data and if we go back to the home path, we will get home once again. So, now it's your turn. Go ahead and add the missing components, go ahead and fill that in. Make sure you understand all of the different pieces of what we just did. Remember, we added the library components to context. We then created a routing mechanism inside of index.js. We, then, referenced the

components that we wrote, and we provided paths that we wanted for them to be loaded on. So, go ahead and fill out the rest of this example, and have some fun.

Video 9 – Bad Bank Context

Next, let's create a context to be shared by our components. We'll start off by adding and creating that context within context.js. Then within index.js, we will create our context provider tag, which will take an initial value that we will assign in a moment. And then that initial value is shared by all of the consuming components, which are the components across the bottom of the page. The syntax looks like the following. We make a call to create context within the React library. And we will call it UserContexts in this case. Once again, the context when we create the tag comes with a provider that will take a value and the components that are consuming that context get access to that value. So, now let's go ahead and write that in the editor. We'll start off by adding to context.js, as mentioned.

And as you can see there, I have pasted my call to create context, and I have assigned it to a variable called 'UserContext'. Next, let's move to index.js and add that tag. I'm going to go ahead and do that here. And I'm going to go ahead and close that tag. Then I'm going to wrap the rest of the routes on the page there. And I'm going to go ahead and collapse my sidebar. Now, I'm going to go ahead and add the value we mentioned. And we're going to make this an expression And inside of it, I'm going to paste an object. And as you can see there, the object that I have pasted in is simply, simply has one property which is 'users:'. And as a 'value=' has an array, and that array has one object, and that object has a potential user. In this case, I have called it ' 'able', email: 'able@mit.edu' ', and I've given it a 'password:' and a 'balance:'.

As you will recall, we're doing a bank. And you could see that this could be a list of users to which you add and remove and that you could also update some of the properties. For example, the balance you could add or remove, withdraw some from that initial amount. Next, let's move to our components. And we're going to start off here with home. And we're going to access that shared context. So, I'm going to go ahead and create a local variable. I'll call it 'ctx' for context. Then I'm going to a function within the React library and this is the function of 'useContext'. And then we're going to reference the context that we created '(UserContext);'. Now, that we have that context, Let's go ahead and write it out to the page. And we're going to use '{JSON.stringify()}' to stringify that object. And we're going to go ahead here and enter '(ctx)'.

And I'm going to go ahead and add a line break just so it's easier to read. Now, we're going to go ahead and do the same for 'CreateAccount'. I'm going to go ahead and replace this. Then I will simply add here 'Create Account'. And one last time for 'AllData'. Okay, so now that we're done, I'm going to go ahead and reload the page. And as you can see there we get in very big letters, all of the shared context, and I'm looking at it here within the home component. If I navigate to CreateAccount, we do not see it. Let me take a look at why and the reason is because we haven't saved the page. If we reload it, then we have access to it, and if we click on AllData, we see the same. So, now it's your turn. Go ahead and add those components and access that context for the remaining components that we see here in the information architecture.

Video 10 – Bad Bank - Styling The Navigation Bar

By this point, we have added our application files and navigation bar, routing in a context to be shared by all of our components. We've done quite a bit, but our application doesn't look very good. And the reason for that is that we have not yet started working on our components. And an easy way to add styling to an application is to use one of the many open-source styling frameworks that exist. And bootstrap, the one I have here on this slide is by far the most used. And so, this is an easy one to use to add to our application. We're going to go ahead and add the style sheet to index.html. And by doing so, we are adding global styles to all of our application. That is that all of the components that we have can make use of what has been defined in the bootstrap project.

So, let's go ahead and take a look at the bootstrap project and make some observations of where we could use this styling first. Now, by looking at our current application on the browser, we can see that it doesn't look very good as mentioned, but the one thing that jumps out is that the navigation bar across the top doesn't really look like a navigation bar. So, let's go ahead and navigate to Bootstrap. And let's go ahead and move to the Documentation. And within it, I'm going to look for NavBar. And once I arrived at the page, you can see here that there's a great deal of functionality and predefined navbars in the syntax that we would use is the one you see before below, the component demonstration.

So, you can see here all of what we would have to use. And you can see here many variants and options that you could include in navigation bars. Now, I have looked at this syntax before and I have selected some basics that we're going to use with our navigation bar. Now, before we get started writing our syntax, we should go ahead and add our styles. And as mentioned, we're simply going to add the reference to the style sheet here. I'm going to go ahead and wrap my code. And you can see there that I'm pointing to the style sheet therefrom bootstrap. Now, by simply doing that, if I reload my page, you'll see that things look quite different. The fonts and just a layout of the page has changed. But this is some default type of styling. We have not yet actively started to use any of the components and we're going to do so with the navigation bar as discussed.

Now, let's go ahead and move to our 'NavBar()'. And let's demonstrate the functionality before we do anything else, let's go ahead and select a random one. I'm going to go ahead and paste it onto this block here. I'm going to go ahead and save it. And I'm going to go ahead and reload the page. Now, as you can see there, we get the navigation bar as expected. And if I go ahead and increase the size of the browser, well, let me go ahead and change the size of the browser. And I'm going to go ahead and increase the size of the browser slightly. And as you can see there, we get the navigation nicely formatted navigation bar as expected. So, now let me go ahead and go back to our earlier code which is much cleaner. Now previously, I went through, I captured or I deselected one of those navigation components template.

And I went ahead and added all of the links that we have here to the component styling and formatting of bootstrap. And I'm going to go ahead and paste all of that here and then I'll make some comments. As you can see here, this is the top of the bar. That line here has to do with the default, the root path. The rest of it has to do with being able to serve to multiple types of screens, the framework can adjust as necessary. And on the bottom part of that component, again, as defined by bootstrap, you can see here that I am adding the rest of the addresses that we had on that navigation bar. So, I have ' "#/CreateAccount/" ', ' "#/login/" ' and ' "#/deposit/" ', and so on, all the way to ' "#/all data/" '. So, I'm going to go ahead and save this and reload the page. And as you can see there, we get our nicely formatted navigation bar.

It says BadBank at the beginning. Then we have Create Account all the way to AllData. Now, if we took a look at behind the scenes on our developer tools, you can see here that we have a warning that says Invalid DOM property `class`. And that is because we really shouldn't be using class, we should be using className. So, I'm going to go ahead and change that. So, I'm going to go ahead and start here at the top and select all of the classes, and then add 'className' to it. Let me scroll down to make sure that I didn't miss any of them. Looks like I did. I'm going to go ahead and add a 'className' here as well. Then I'm going to go ahead and reload the page, open up developer tools and you can see that that warning is now gone.

So, we have the exact same thing that we had before, the paths, however, I am now wrapping them in a component that I got from bootstrap. And so, we have loaded the styles, we have loaded the syntax and we have wrapped our previously neat set of links into a much bigger component that gives us the look and feel that we were looking for. Now, it's your turn. Navigate to bootstrap, select some components that you'd like to add to the pages. You can add them at any place you like. The exercise is mostly to get comfortable with taking components from bootstrap and adding them, importing them into your components in your application. There's a lot that can be done. Take a couple of them and try them out and see how you can import good-looking components into your application.

Video 11 – Bad Bank - Styling The Bootstrap Card

Now that we have a style navigation bar, let's turn our attention to the rest of the components. And for that, we're going to be using a Bootstrap style called card. The initial example that we gave of the application had a component that is the default component at home, the root component and it was wrapped in a card. And you can see there what a card looks like. Now, we're going to be adding a new component, and this will be the card component. We will add that to context.js, and all of the rest of the components, as opposed to having to do their own implementation of the card syntax, will be able to reference that card component and be able to pass properties for those features that are relevant to that component. And so, this is the generic component within Bootstrap syntax.

And we're going to be making properties of a few of the syntax components on this component from Bootstrap. So, we will be passing additional options for the 'class'. We will be passing in a

title for the ' "card-header" '. We will do the same thing for ' "card-title" ', which you can think of as a subtitle inside of the body of the card. And we will do the same thing with other parts of the card. We will also add some additional properties which are more relevant when it comes to what we are trying to do. Now, there are some colors that Bootstraps define, and I wanted to write down here what their naming is. And you can see there that blue is primary, the second one is secondary, and so on. And as we get to the end of this lesson, we will demonstrate at least one of those colors.

So, now let's go ahead and move to the editor. And we will start out by opening the 'context.js' file. And inside of here, we're going to write a new card component. And we'll start off by writing our 'function', then the 'return' syntax. And within it, you can see I'm starting to define the syntax for the 'Card'. This is HTML that I have taken from Bootstrap and I am now importing into our application. You can see there that I have changed the name of 'classes()' to 'className' as it's required in this environment. And that I have a couple of expression. The first one is making a call to 'classes()', which we will define shortly. And the other one is an expression for the 'style'. You'll note there that I'm passing in an object. And that is because I want each component to have a maximum width of 18.

Now, the next thing that I'm going to do is, I'm going to add the ' "card-header" '. And you can see there that that is going to take a property of 'header' that the component that is using this card can pass in. Likewise for the body. I'm going to start to add a number of properties. In these properties, I'm going to check for their existence. If they exist, they will be added to the page, otherwise, nothing will be added. Then next, the same thing happens for 'text'. And then the 'body' property, where each of the components can add the body of their content. I will do the same thing then for ' 'createStatus' '. And this is something that once we start to interact with each of these components; that is, we are creating accounts, we're logging in, we're depositing money can be used to modify and update the UI.

Now, as mentioned, we're going to have an additional 'class' that is going to work with setting up the 'classes()' for the styling that we need. And you can see here the syntax that I'm going to use. I'm going to check to see if a background color 'bgcolor' was set. If one was set, I will choose it. Otherwise, I will leave it empty and go with the default. The same thing is true for the text 'txt'. I will select the 'txtcolor' that I'm going to use, and mainly that is the default or 'text -' or white text. The next thing that I'm going to be using has to do with the previous values that have been defined for the background 'bg' and text 'txt'. So, with that in mind, let's go ahead and save this file, and let's go ahead and write a first component with this new card component.

And so, we're going to write 'home.js' first. I'm going to go ahead and move to that now. Let's go ahead and open that up. And as you can see here, we have a very simple component. At this point, I'm simply going to remove what I have here. And I'm going to add some new syntax. And that syntax is going to correspond to what we just wrote in the card component. And so, I'm going to start off by creating my '< Card >' tag here. And actually, I'm going to terminate the '< Card >' here, and then add the attributes which will be passed in as 'props'. So, the very first thing that I'm going to, that I'm going to set is the 'txtcolor'. And the 'txtcolor' is going to be ' 'black' ', in this

case. And I'm going to change here to double-quotes, which is the traditional type of quotes that are used for attributes. Then I'm going to set the 'header'.

And I'm going to call this ' "BadBank Landing Page" '. The next thing that I'm going to enter is the 'title'. And for the 'title', I'm going to write ' "Welcome to the bank" '. Then I'm going to set some 'text', and I'm simply going to enter all of these to illustrate everything that can be added. And I'm going to say here, ' "You can use this bank" '. And then the 'body' of the card. And for the 'body' of the card, I'm going to go ahead and evaluate an expression. And this expression will simply be an image. And you can see there that I'm setting the name of the file ' "bank.png" ', and we will add that in a minute. Then the 'className' and ' "Responsive image" ' is just an alternative, in case, the image cannot be displayed.

And the 'className' is taken once again from Bootstrap. So, as you can see, we have been able to add a card with pretty clean syntax as opposed to the more busy type of syntax that is required when you're writing all of the HTML that Bootstrap requires. So, let's go ahead and add our image file. And as you can see there, we now have it in our directory. And I'm going to go ahead and reload the page. I'm going to move to the root. And as you can see there as expected, we get our card. And there's a little bit more formatting to be done. But you can see there that now all of the application looks styled. Now, if we wanted to change the color, the background, we could go ahead and add an additional property and this would be 'bgcolor'. And we would set ' "primary" ' here. We're going to go ahead and save, reload the page.

And as you can see there, everything now is blue. And I believe that if I simply enter here ' "white" ', we should be able to get white text. And you can see there that that looks pretty different. And we have not done that much different. So, as you can tell, we have a working card that takes a number of properties and that allows us to format our components in a pretty easy way. So, now it's your turn. Go ahead and add the new card component. Make sure you can wrap your elements. Go ahead and try a few variations. There is, there are a few moving pieces here, but as you can tell, once you get a nice template that you add into a component, then you can leverage that from the others. This is a nice practice to get into.

Video 12 – Bad Bank - Create Account

We have done quite a bit. We designed our application, created our navigation, did routing, context, and added styling to our components. Now, we're going to create an interactive forum where we will take information from the user update our state, and update our UI. Now, this can be pretty involved. So, let's overview the number of steps that we're going to write code for. We'll start off by writing our state variables. We will have a status to give information back to the user depending on actions. We will have state for name, email, password, and the UserContext that we have been using. Now, depending on what state we are encountering, we will show one form or another. And this will be written in our logic, this option between these two forms.

In either case, we will have a number of fields which correspond to the data that we have shown to create an account. And then a button that will fire an event. And so, we will have these two

states depending on whether the users creating an additional user or we are adding one more. That is, the action has already been taken to create the user. Now, we will handle these events by a number of functions. We will have a handleCreate. We will have some validation, and some ability to clear the form if we're coming back to add an additional user. So, at a pretty high level, the form that we have is the first one we took a look at where we have a number of fields, and then we have a button.

And all of the rest that we just took a look are to handle the information that's being given to us, what we should show for the UI and how to update our state, our list of users. So, now let's go ahead and get writing our code. As you can see here, I am in the 'createaccount.js' file. And I'm going to start with the state information. We're going to create here a series of these. And as you can see, the very first one is the one that sets the '[show,' variable. The second one is the '[status,', the third one is the '[name,'. The next one is '[email,', then we have '[password,'. And the last thing that we do is going to handle on the '(UserContext);'. This is what we have been doing for some time. Next, let's write our JSX. And inside of it, we're going to add the logic or the '< Card >'.

And inside of it, we're going to set our properties that we're going to pass into the card. And then the body which will hold our forms. So, inside of it, we're first going to write the properties that we're going to pass. We want our background to be ' "primary" ' color, then the 'header' of that form of that component is going to be ' "Create Account" ', then the 'status' is something that we're going to pass based on the actions that are taken. And finally, we're going to write our 'body'. Inside of the body, we're going to have a ternary expression. And as you can see there, depending on what the value of 'show' is, which is defined on line '1'. And we start off as true. We will show one form or the other. So, here we'll start writing our code.

And as you can see there, I have put in place holders that is empty tags to be able to differentiate the two placeholders that we have for the two forms. Now, I'm going to go ahead and start writing the first one. I'll start off here with a label for the 'Name'. Then I'll write my first field. And as you can see there, I have a 'type' of ' "input" '. Then the 'className' that I'm using comes from the styling in Bootstrap. I have an 'id', I have a 'placeholder', then I have a 'value' that will be evaluated. This is another one of those state, of the state variables that we have within the component. And I have an 'onChange' event that is simply going to set that value. We will need that to be able to handle the event once we get to the section, that will be a button that will fire that event. The next thing that we're going to add is another field or 'Email', in this case.

And as you can see there, it's very similar to the previous one. Now, we'll have another one for the 'Password'. Very similar, the styling, the 'value' is evaluating, in this case, yet another variable that we have. We're handling the 'onChange', and we're setting the password using the definitions that we did in the first lines. And lastly, we will have our 'button'. And as you can see here, that 'button' has an 'onClick' event to a function called '{handleCreate}' that we will write shortly. But before we do that, let's go ahead and write our other form. And as you can see there, we have a label there of 'Success', which is the form that we will show when this form is submitted. And then we will give the user a chance to create another user. So, that will be the chance for them to click on add another user.

At that point, we will call a function called '{clearForm}', which will clear the form so new values can be filled in. And let's go ahead and move now to the functions that will handle those events. And I'm going to start off here with 'clearForm()' event. This is a simple one to use. It will simply clear all of the values and it will 'setShow' to '(true);' so that we can once again see the form fields in the first option of those two forms. Now, I'm going to go ahead and write the validation. And this is a function that we will use to validate the fields. And there's a lot that you can do with validation. In this case, we're simply checking for an empty field. There's a lot more, and we can leave that to you to explore additional things that you might want to validate. We will simulate here submitting a form and we will have a delay just simply for the purposes of illustrating a typical type of interaction.

And we will then, depending on whether the field was validated or not, we will 'return' 'false' or 'true'. Now, we will finally handle the create, the create function. And we will write to the 'console' the 'name' and 'email' simply to be able to troubleshoot to see if the function is getting the values that we think it's getting. And then we will go through validation, so the field. We will 'validate' the 'name,', 'email,', and the 'password,'. Then if all of that is successful, we will push the new user into our user component. And the last thing that we will do is 'setShow' to '(false);' to be able to hide that initial form that we see and give the option to add another user, as we showed when we were handling the two cases for the two forms. To overview once more, what we have done is we have created our state variables.

We then went ahead and created are two forms that we're going to show depending on the value of the variable show. Then we added that second, the two forms that we're going to handle that. And finally, we added our event handling functions. Now, let's go ahead and save our form. Reload our page. Navigate over to 'Create Account', and as you can see there, we get what we expected a 'Create Account' form. I'm going to go ahead and enter a new user. I will call him 'peter'. And I will enter 'peter@mit.edu'. I will enter a password. I will make the password 'secret'. I'm going to go ahead and open up my developer tools so we can see the feedback. I'm going to then go ahead and press on the button to 'Create Account'.

And as you can see there, the information that we get back is what we expected. 'peter peter@mit.edu' and 'secret' for a password. That is the place where we're writing to the console. And you can see there that we're now seeing the second form. The one that we defined here. And that if we press on 'Add another account', the 'clearForm()' will be fired. We will reset the values of name, e-mail, and password. And we will 'setShow' to '(true);' for our first form. This one here. This one that starts here in this place. And so, I'm going to go ahead and do that now. And as you can see there, we get the form back. And if we were to take a look at 'All Data', we'll see here that we get the information back that we expected. Now, it's your turn. There's a lot to keep track of.

Your updating state, your updating UI, your conditionally showing forms, your handling events, your updating values, your validating field. There's a lot to keep track of. And this is why forms can be challenging. So, now it's your turn. Go ahead and replicate the work that we have done. Try to see if you can organize it in a better way. When it comes to design, there are many ways

in which you could achieve the same goals. Now, in addition to that, we want you to do the rest of the forms that are represented there on the navigation bar, that is the login the deposit, the withdraw, and a balance. This will give you a good opportunity for a new set of conditions that you're trying to achieve.

The login, can you match the information that the user provided to that what you have in the useState. Similarly, can you add to the balance of the user when it comes to the deposit? Same thing with withdraw. And ultimately, balance, being able to check the balance of any one user by their given email. And so, this should give you some practice, and this is very useful because many applications run into the same type of conditions. And this will make you experience that in a concentrated way. And there's a lot that could be done. You could take this a lot further and do a great deal more, but ultimately, make sure you fully engage into the application because the complexities, the challenges that you will run into will help you quite a bit down the road.

Video 13 – Bad Bank Challenge

One of the interesting things that happens when you finish a large application is that you start to see opportunities on how to redesign your application. How to create something better, that it's cleaner, that it's easier to maintain, that it's easier to author. And the same thing is true when it comes to Bad Bank. How could you improve the work that we have done? One of the first things that comes to mind when you take a look at the code is that we're repeating a lot of the work that we do in forms in every single component; this is wasteful. How could we do it differently?

Perhaps you could create a form component that all of the other components the createaccount.js, login.js, deposit.js, and so on, leverage. That is that they use that and simply pass properties. Could that greatly simplify your application? I did an initial exercise to explore some of the possibilities from the brute force approach, say that we initially took, and yes, you can. And so, here is a hint on how your createaccount that has quite a bit of code could look if we took that approach. And as you can see here, it is a much cleaner application.

And instead of handling all of the events, we just handle some of them. And we could pass those on to that form component for all of these components that we have within Bad Bank. And you can see there that we could create a much cleaner design. So, how would you improve it? This is the challenge that we're putting before you. And this is a good way to think about refactoring, about redesigning, about thinking about the capabilities that you have on the platform and how you could do things different. So, this is your turn to shine, to flex your muscles, to move into a cleaner application.

Video 14 – Growth Mindset

What are you good at? Many of us have a pretty clear idea. We might be great mathematicians, but not such good musicians. Or the other way around, we might be great artists but terrible

mathematicians. Now, imagine for a moment that you could be good at everything. It seems like it's not possible, right? Many of us know from a pretty early age what we're good at. Well, it turns out that we can be good at everything. And what limits us many times. It's what's referred to as a fixed mindset versus a growth mindset. If you think you can do or you can learn anything that is a growth mindset. That is that any temporary or you view any type of obstacle that you run into, any type of difficulty that you have as something temporary and to be overcome.

And that mindset where we have fixed abilities in we're either good or not at something is something that decades of research have shown to be false. So, if you're not good at something, it simply means you need to dedicate more time to it. And a lot of the research shows as well that most of the time when we fail at something is that we're trying to do too much. Think of a complicated choreography and dance. If you have to master or you have to replicate 30 steps, that might be pretty difficult to do. But if you have to do one, that's not so hard. Even myself, who I consider myself not to be a good dancer, which shows some fixed mindset there can do it, right? And so, this is a very valuable thing to understand because it simply illustrates our tremendous capabilities to be able to learn anything we like.