Shopping Cart Exercise

Video Transcript

Video 1 – Introduction To The Shopping Cart Exercise

So, who would have thought a shopping cart could be so complicated? In this exercise, we're going to develop a complete app. And it's going to consist of a shopping cart. There's going to be a store that we're shopping in and we're going to checkout. Now, we're also going to replenish the store when our stock runs low by hitting a remote database. We're going to actually set up a database using Strapi. It's going to give us an application program interface, an API that we can hit and retrieve stock. We could actually put new things into the database if we wanted to. And we'll show you using Postman to actually hit Strapi, and make new requests and push new data into the Strapi database. But the focus is on the shopping cart, on our React app.

And in that, we'll be wheeling our cart through our shop. The shelves will be stocked with goods, we'll be taking goods from the shelves and putting them into our shopping cart. And so, the stock in the store will decrease, but the items in our shopping cart will increase. We're going to be keeping check of the checkout total as we go low so that we know how much we've spent. So, we'll have three parts to this. We'll have the store that's keeping track of the stock, we'll have our cart that's keeping track of what's in it. And we'll have the checkout total that we need to provide when we check out. There's some design decisions which I made. I decided not to have separate web components, but to roll this into a single web component. Maybe you will make a different design decision.

Video 2 – Create Strapi Product Database

So, we've already built our cart user interface. Now, what we want to do is to connect it to a real database. We're going to host the data in Strapi. And this is a product allows us to create an API that we can hit with our fetch commands from the cart. So, we've already done fetching of data. Now, we're going to do it to a real database. Okay. So, let's install Strapi and get moving, okay. So, you can find more information about Strapi if you go to strapi.io. And this will give you a good introduction as a quick guide here how to get Strapi running in three minutes. So, I'll take you through those steps now. So, first, create a directory. We need to be in an empty directory. So, bring up a terminal window, navigate to where you want to put the directory, then say 'mkdir' and the name of the directory.

So now, we can 'cd' into 'cartDB', which is what I've called it. And now, we can build Strapi. And to do that, we need to do 'npx create-strapi-app'. And now, we give it the name of the directory that we want to create. So, I'm just going to call it 'cartDB' again. And then we want to have a quickstart, so '- -quickstart'. And it's going to start to build it. So now, it's doing quite a fair amount of work here. It's building using Webpack. And you can see that we can run these various commands to start. 'npm run start', 'npm run develop'. And this is the one we're going to use in the future. And now, Strapi has stood up a server and we'll see that it's running on localhost:1337.

And it's asking us to fill in our name so you can fill in your name. I've already done this, I believe. And I got my password, and confirmation of the password. And then you can start. So, we can now begin to create content. So now, we have Strapi up and running. We need to build content. So, we're going to build products. So, if you remember, these are our products. So, we've got 'name:', 'country:', 'cost:' and 'instock:'. Let's go and build a content type. So, we click on Content-Types Builder and we need to create a new collection type. So, this will be our products and our collection type. The name will be the single one product. Okay. And what do we want? Well, we want Text. That'll be name. So, that's the name of our product.

Now, we want to have the country, that's also Text. So, country. And we've got cost, which will be a Number. And so, cost and our number format. We'll go with decimal. Let's just go with integer, for now, that's fine. And instock, we need the Number instock. So, we've got instock. Number format, integer. Okay. So now, we need to save that. Remember to save it, okay. So that's saved, and now we'll see we've got products added here. So, let's go to the Products, and now we can enter new products. So, Name, let's actually go through and put what we had here. So, let's put in Apples. Country, Italy. Cost, 3. Instock, let's put 10. So, we save that. So now, you see we've got that item, add another product.

So, Oranges. Country, Spain. Cost, 4 and Instock, 3. Save that. So, you see we've got them already. And now, Beans. Country, usa. Cost, 2. Instock, 8. And we'll add one more. Last one is Country, usa. We did it in small letters so it's fine. Instock, 8. Actually, let's make that caps. Let me go back and do caps for the other one. Save. So, I think we had, this one is small. We can edit it. So, here. Okay. So, now we have our products set up. Let's see if we can go and get them. So, to hit our products. Strapi automatically sets up an API and it's going to be at this server, and it's going to be called Products. So, we can hit it first in our browser. Let's bring our browser up. So, we, in our browser. So, if we go to localhost and we want, so, localhost, and then just products.

And you see we've got an error. That's because we need to give permissions. So, let's go to, so we need to go to Roles & Permissions. And at the moment, let's take a look at this. This one we want to give permissions to Public, and here's our product. And we don't have any permission sets, so we need to select them all. I'm going to allow Public to hit any of these. So, count, find, create, delete. Okay, so now, we've got those permissions. Now, let's hit. Here we go. So, we've hit our API. And you can see it brings back our products. It has a few extra fields like creation date, username, you know the firstname. That's good. That's excellent Actually. So, that's working. Okay, so now, why don't you go and install Strapi and create your own products. And then we'll see if we can incorporate them into my shopping cart.

Video 3 – Install Postman

Okay. So, let's install Postman. So, you go to postman.com/downloads/ and it has downloads for Windows and Mac and we click on Download. And we're going to get a zip file. And you'll unzip the zip file, and go through the installation. And when we've finished, we should have it in our applications folder. So, if we look at Postman here, then we can double-click and bring up the app. So, what Postman does, it allows us to send, GET and POST messages to any URL. So, for example, here, we could send this to Strapi. So, for example, we had products. And if we do a Send, then you'll see we get back a list of the products and we can scroll through. So, this is really useful.

We can POST, for example, let's suppose we want to POST something to Strapi as well as doing GETS. We can do POSTS And in this case, we need to have a Header. And so, the Content-Type needs to be application/json. And in the Body, we need to put the object that we're sending. So, we need to put it in JSON format. So, these have to be in quotes. And "nuts", I need in quotes, these others are numbers. So, let's send it. Okay, and we get back that it's got an "id": of 5, "name": "nuts", "instock": 3. So, it's overwritten the "id". to give it a unique "id", and it's putting other data as well. So, to bring up Strapi, we just do localhost:1337 and LOG IN. Okay, and if we look at our Products now, we'll see that we've got nuts.

We didn't specify a country, but we've got the cost, et cetera. So, we can now talk to Strapi as a database through this API. So, going back to Postman, we can now look at, if we click on this Code, it will show us how to make that request in different formats. So, we've used Fetch if you remember to make the request. And this shows you the format that you need to use to make a request. So, you'll need the Headers. And then these would be the request options and specifying the JSON format. And also here then is the fetch request. So, if you want to make it, for example, with cURL, then also it'll show you. So, cURL can also make requests to URLs, and it can GET and POST. And this shows you the same request that we made in Postman in cURL. Okay. So, that's another facility the Postman gives you and we'll use it more in debugging code, for example. So, that's Postman.

Video 4 – Using Postman And Express Web Server

So, today we're going to look at the Express web server. And we're going to explore how we use roots to control which web page you're getting. So, the first thing is go to your directory and do 'git clone', this repository. Okay? And this will give you, then these files. And we need to first run 'npm install'. So, the 'package.json' will install Express for us. So, 'npm install', that's going to build everything for us. And now, we can run this web server. And the webserver is called 'contacts.js', this is an arbitrary name. I just happen to call it 'contacts.js'. We'll see what's in that file now. But I just want to show you that it sets up a server 'Running on port 3000'. So, if I go to a browser, so here's my browser. And if I go to 'localhost:3000', you'll see it says 'Routes: try POST to /contact and GET /contacts' to see what contacts we've generated.

So, this is a little webserver that's going to allow us to set up contacts and display which ones we've already got. Okay, so let's go take a look at the code. So, here's the code. So, this is 'package.json', and we just installed ' "express" '. We could have done it by hand by typing 'npm install express', and this is version ' "^4.17.1" '. Now, let's look how we program this. So, here we are requiring '("express");', that library. And we're going to set up our 'app' to be 'express();'. And we're going to use JSON. Now, when I say use JSON, we're going to move data around in the JSON format. Now, I'm setting up an array to store the 'contacts'. Now, we should really use a database, but we're going to use a local array here because I just want you to see it and have control of everything. We're going to populate 'contacts' with 'contacts'.

So, for example, we have a name and an e-mail that will specify a person. Now, here, we specify that if we get an HTTP get request, and it just does localhost 3000. Then this is the default route. It will execute this. And this 'res.send' says, I'm going to send you back this HTML page. And all we've got in our HTML pages is '< h1 >' tag with 'Routes:' 'try, POST', 'and GET'. Okay? So, that's the default route. Now, we've also got another route here. And this is a 'get' HTTP 'get', again. But if we have 'localhost:3000"/contacts" ', then it will come here. And what I'm doing here is I'm sending back in the response 'res'. So, the request 'req' really doesn't have anything in it here. We've just hit this route. But we're going to send back in JSON format all the contacts that we have, already.

Now, at the beginning, we won't have any. And to 'add a contact', we do a 'post'. So, we do an HTTP 'post'. Now, to do these, we, the browse can only do 'gets', unless we've got a special page there that will do a '.post' for us, but in the URL we can only do 'get'. So, we need something to do a Post for us, and we're going to use Postman. So, I'll show you how to install Postman, and let's use it now to hit these various routes here. So, I've installed Postman, and I've been using it, obviously. But now, let's do a 'GET'. So, here we can choose POSTs or GETs, and these other ones as well, but we're just going to use 'POST' and 'GET'. But let's do a 'GET', and let's do it on 'localhost:3000'. That's our server. Now, we need to make sure that our server is up and running. Let's make sure that we've got our server.

Yes, it's up and running, so we're good there. Now, let's send a message to it. And in the 'Body' I'm going to put anything because I'm just going to hit '3000'. And the 'Headers', I'm not going to have anything in those either. And we don't need any authorizations yet. But we'll be coming back because usually the messages, the HTTP requests will need to be authorized. And let's hit '3000', and it just returns to us this. So, we need to do a 'POST' to '/contact'. Let's just make sure that we got that right. So, we're going to do a post to this contact, and we need to put in the 'name,' and the 'email:'. But that's going to be in the body of the message. So, let's go back to Postman. And now, we're going to do a 'POST'.

And in the 'Body' of the message, we need to put, well, we need to put the ' "name" '. So, let's enter. I'm going to enter my name for now, ' "john" '. And we need to put the email address. So, I'll put ' "email" ', and my email address. Okay? And I'm putting it in JSON. This is an old JSON object. So, that goes in the 'Body'. We don't have anything in the 'Headers', and we don't have anything in the authorization. But now, let's send that 'POST'. And we'll watch down here. So, we'll watch down here for the result. So, here's the post. And we'll see that it echoes back to us ' "name" ', ' "email" '. Now, let's go and look at 'contacts'. Let's go back to the 'GET'. Now, it keeps a list over here of all the previous ones that you've done.

So, it's easy to recall this one. Hit the route 'contacts'. Now, that's a 'GET' on 'contacts'. And it returns to us the array. And all we got in it at the moment is just my one contact. Let's put a couple more contacts in. So, we'll go back to the 'POST', go back to the body. And here, let's put in ' "fred" ', and send that. Yup, so ' "fred" ' has been put in. Let's now post another one. Let's put in ' "anne" '. Okay, so ' "anne" ' is being put in. Now, let's put in ' "abel" '. Okay, okay. So, now let's go back and do a 'GET' on the 'contacts'. Let's have a look, 'GET' on 'contacts'. And we should be able to get all of them back. Let's have a look here. Yes, ' "fred" ', myself with ' "fred" ', ' "anne" ', and ' "abel" '. We've got everyone. So, you see that by using these routes, we can control which function is fired in the webserver.

We've only got a couple of them up at the moment. But we can build this out so that we could, for example, put this into a database. So, one of the nice things about Postman is that we can take a look, for example, at our last 'POST'. So, we posted, and in the 'Body' of the message we put ' "abel" '. So, that was in JSON format. We didn't have any 'Headers'. But now, we can, by going to this 'Code' button, we can look at what that request looked like in various formats. So, let's take a look at, for example, the 'HTTP'. So, here was the 'POST', and you see that the data was attached like this. Now, if we go to 'cURL', which could have made the same request, then you'd see that we have '--data-raw', and then ' "abel" ' here. It's a 'POST' to this URL and the '--header' has got this format.

So, this shows us, particularly, what the various 'HTTP' requests are like. Let's just take a look at a 'GET'. So, here we have a 'GET' for 'contacts'. In the 'Body', we didn't have anything. This was our output down there. So, let's just take a look. So, here, it's just a straightforward 'GET' to 'contacts' at this URL, this hostname. So, if we've made that with a 'cURL' command, then that's what it would've looked like. So, we interacted here with a web server through Postman mainly, but we could also do it through cURL or other methods of issuing these HTTP requests. Okay, so that's Express. And we'll be using a lot more of Postman to debug our APIs, and routes are very important to understand. So, we'll see them again.

Video 5 – Shopping Cart Exercise - Refactor Introduction

So, in this exercise, I want you to take the starting code and make it do the following. So, first, I want you to change the pictures so that they come from Picsum. I'll put the link to that site, that you can pick up random images from Picsum. So, if I load again, you'll see the images are different. So, I just want you to do that first. Next, I want you not only to list out the price of the Apples but how many are in stock? We want to keep track of how many other people can get if we put some in our Cart. How many are in stock? So that, for example, now if I choose to put Apples into the Cart, they now have been removed from the shelf in the store and they're in your Cart.

So, we've only got nine left in the store for other people. I want you also to note here how much it's costing. So, when we CheckOut, we'll know before we click on the button, exactly how much we've got in ours. So, now Oranges, and it's going to cost us seven if we CheckOut. Also, make sure if we go to zero, that we can't put any more Oranges into our basket. Now, we can delete, we know by clicking on the accordion. And now, if I click again, notice that that Orange will be taken out of the CheckOut, and be put back on the shelf, and so, we'll have in stock rather instead of zero, we'll have one.

And if we delete another Orange. So, here notice the stock goes to two, and we've moved it from here. So, I want you to put in that coordination, and also the ability to read stock. So, here we've added some back into the stock and make sure that they work as well, that if I choose, for example, let's choose some Beans here. Then they go into the Cart, and also into the CheckOut and we keep track of the total. So, make sure that everything's sinking up. Notice if I take it out of the Cart, then it comes out of the CheckOut as well. Okay, so that's the exercise.

Video 6 – Shopping Cart Exercise - Refactor Restocking Functionality

So, we're familiar with the shopping Cart and I'm giving you now the solution for the previous one, where we have a number of products and we want to be able to put them into the Cart. And possibly we want to take them out of the Cart by deleting them. We'd like to have the price, keep track. So, if I remove the Beans, for example, price will change by 2 and down to 7. Now, what I'd like to do is to have this be restocked. So, if we run out of these Apples, now at the moment, we're not decrementing the number that we have if these are bought, for example, if we CheckOut these Apples then the number in stock doesn't go down. So, that's one thing we need to do is to make sure that these numbers change as we buy things.

Now, we could decide to decrement them when they go into the Cart. So, let's do it when the items are put in the Cart, we need to decrement them over the product list. Now, the next thing is to, we want to go to the Strapi database. So, that's running on localhost:1337. That's port 1337. And we go and get the products. So, I've done this for you that when we click there, it goes out and gets the products. So, here we've got three more. And they're just copies of these up here that we've got. And we started actually at Oranges, now we start with Apples. Okay. So, that's the starting point for you. Let's take a look at this button-down here because this is the button that does the work to go and get the data from Strapi. I'm going to go and fetch the data.

Let's take a look at the code. So, here we are now in Products. So, this is our web component, 'Products'. Let's go and look at where it's rendering. So, down here, this is where it's rendering everything. It's rendering our 'Product List'. So, we need to make sure that if we restock that this '{list}' has the new products, this is our '{cartList}' that will stay the same. And here we're on 'CheckOut'. So, that should be fine too. Now, we need to add a form here to go to get restock of the products. So, here we've got a '< form' and we've got our '< input', and we've got our ' "submit" ' and 'onSubmit' here of the '< form'. We're calling 'restockProducts'. We've also got a write statement, 'console.log', and we need to prevent this being called all the time.

We only one, we called once when we clicked on this submit. So, let's take a look at 'restockProducts'. You can see it's going out to the Strapi database. And we've got this '${query}' which should be products. So, let's take a look at 'restockProducts'. So, that's going to be here. So, it's taking the '(url)'. And we're assuming that it's correct. And now, it's doing a 'doFetch'. So, this is going to set that '(url);'. So, it's going to use the useState. And when the useState has changed, it's going to trigger doing the actual Fetch, which we saw last time. Now, we need to arrange so that what it gets comes back as this 'data'. So, we need to find out where 'data' is set, because this needs to be the array of new products.

Now, what we do is we pick out each item of the products. Now, because they're coming from the Strapi database, the items have some extra fields, but we're only interested in the '{name, country, cost,' and 'instock}'. So, we destructure 'item' and put it into this object. And now, we return that object. So, that's being returned here. So, 'newItems' now is a list or rather an array of these objects. Now, we want to add them to 'setItems'. So, we spread the present '([. . .items,' and add in the '. . .newItems]);'. Okay? So, that's going to change 'setItems'. And so, the items are going to be changed and the next time we render the list of items up here, we create that 'list'. You'll see that. So, in that 'list', here is what's being created.

We have an '< Image', we have a '< Button' and a ' "submit" '. So, that's listing the products out. And at the moment, what I'm doing is I'm just repeating the images that are there. So, this may get a little mixed up. We might have the wrong image against the new products. I'm using the four images that we've already got. What I'm going to ask you to do is to go out actually to do this, pick some and pick out photos from there. Now, they won't be your fruits, but they will be random images, and you can set the size of the image. So, I'm setting the id here to be 'index +' some number. You can choose, whatever you want. It'll get certain, a range of images. But that they are going to be this size, 50 wide, and 50 high. Okay. So, that will render everything as you saw.

And we should be good from there. And we've seen how to do the Fetch. Remember that's up here, where we got 'useEffect'. That's going to be called because we've changed the URL. Remember that It's tracking the URLs here. And so, it's kind of go fetch the data. And where is it being returned? Well, it's being returned there in 'state'. And where do we call this? Well, we're calling this 'useDataApi' in Products. So, we need to go down to Products. And we'll see here, we're calling that 'useDataApi'. So, this is that fetching it, triggering it. And here is the 'data' coming back. It's part of this object. So, we pick it out, we have access to it now 'data', whatever we put here will be the data that's coming back. And we need to make sure we understand the structure. It turns out we do. So, when we're rendering it we're okay. So, that's getting the data. And now, we'll do the exercise.

Video 7 – Shopping Cart Exercise - Refactor Solution

So, we've seen our Shopping Cart in action. We've added some functionality to synchronize putting items into the Cart and taking them out. So, let's take a look at the solution that I came up with. You may have come up with a different solution, but as long as everything works, then we'll be fine. So, let's begin. Let's take a look at the code. So, first, we need to be able to take things off the shelf and put them in the Cart. So, here I could have stock of nine left. I've already put one in the Cart. Notice that already in the CheckOut, it is showing $3 and that we've got one of Apples. Let's put another. So, it goes here and it's in the CheckOut at $6. We realize that we don't want that many Apples. Then we can delete it from our Cart. Notice that the stock is going to go back onto the shelf.

That goes to nine. Notice here, it was taken out of the Cart, and also the total goes down. So, it's the synchronization, that's the most difficult part of this. We also needed to get the images, which is fairly a step straightforward, and we need it to be able to restock. So, obviously, we need to put a form here and have a submit so we can go out to fetch the data from Strapi. Okay, let's take a look at the code. So, here's our code. We notice that we've got a component, 'Products', and we also got to 'cart' component. So, here's 'Products' component starts here. And then our 'cart' is quite a short one here. The 'cart' takes care of the 'Accordion'. We've got an ability to, we're using 'DataApi' to take care of our state. Inside here, we've got 'useEffect', which when we're getting, we're fetching data from the warehouse that we change the URL, and that will automatically fire a 'fetch' from the database and we're pointing at 'Strapi' database.

Okay, let's take a look at where the coordination happens. So, when we add to the Cart, we need to coordinate and make sure that it's also added to the 'checkOut'. So, let's take a look at this. So, here we're using the 'target.name;'. So, this event is being passed to us when we click on the submit for adding to the Cart. So, now what we need to do is figure out which item was clicked on. So, we're doing that with a 'filter'. Now, remember, 'filter' always returns an array. Even though we're expecting just one item because we're matching the 'item.name' with the 'name' that we've got from where we clicked and will only ever get one 'item' by that. Well, at least we should. Then we need to get the first in the array and look at 'instock'. Here we're checking that we've got something 'instock' that we've got over '0' items.

If not, we're going to return. So, if you tried to add something to the Cart, and there aren't any on the shelf, then you can't do it. If they're on the shelf, then now we're decreasing the number of items on the shelf by one, and we're adding that item to the 'cart'. So, remember here that we're spreading and we're spreading this, and we're putting it into a new array. Remember when we're setting on 'useState' that we should always treat the thing that's being set as immutable. So, we've got to give it something new. So, here we're creating a totally new array and spreading 'cart', what's already in there, and adding that item in. Okay, so that's 'addToCart'. Let's take a look at 'deleteCart'. So, this is going to be called when we're deleting an item. And here, we've got the 'Index' of the 'Item'.

Now, remember it's the index of the item in the Cart because we've also got products on the shelf that are items as well if you like. But we're not going to deal with the Cart one, so this is fine. Now, we're going to find out which one we're deleting. So, the 'newCart' will have all the items except when 'delIndex != i);', so that one won't be put into the 'newCart'. So, okay, now we've got the 'newCart'. Now, we need to know which item was deleted. Because now we need to go to the 'items.map' and increase that by '1', the number of items. So, we need to find it first in the items.

This index won't give it to us. We've got to calculate it. So, we go through map all the items. We check if the 'item.name = target[0]'.

Remember again, 'target' we get from a 'filter', so it's an array. We need to get the first element to the array and we check the names, okay? And then we put it back on the shelf. So, that's what puts it back on the shelf. And we 'setCart(newCart);' because we've got one less item in the Cart, and we've got one more item now back on the shelves so that this, these to do all the work. So, basically, that's most of it. We can look at getting the list where we get the photos. Here, I'm going to pick some '.photos'. The id just allows us to go to different sections of their photos. I chose. There's nothing special about '1049;', and we're in, we're getting different ones by increasing 'index'. So, 'n;', as we loop around the map over the items, we're getting a different photo for each item. Now, each time we do it, we're going to get a different photo.

So, this is not ideal. I'm just doing it as an exercise. So, here now we're controlling the width of the photo, and we're making it a 'roundedCircle'. That's just per my choice. Okay. So, that's the main things that, here's the 'finalList'. So, this has to do with 'checkOut'. Here we're getting the 'total' price in CheckOut. So, we go to the 'cart' and we map all the items in the Cart, and basically, get the total here with 'checkOut'. That's going to be a reduce to get that total. And then this is the 'final' list of items in the Cart which we're listing out at CheckOut. And here, I'm returning an object with '{ final,' and 'total };', that was just my convenience there's nothing special about that. So, take a look at this code. You can begin to see how much work you have to do keeping track of items.

Remember that when I get items from the warehouse, they may be different. For example, Apples to the ones I've got on the shelf. So, I'd really need to check the price and where they came from. Most products have a skew, a stock-keeping unit. I check the skews and only add them to the ones that were already on the shelf if they were exactly the same. But if they're Apples that are different, they'll have a different skew. So, I need to list them differently and be able to put those into the Cart separately. As you know, with Apples and Tomatoes, you go to the supermarket and there are lots of kinds of Tomatoes, Red Tomatoes, Green Tomatoes, Yellow Tomatoes, etc. So, keeping track of all this gets quite complicated. But we've got a long way to understanding the problem here. So, congratulations if you got this far.