**Lecture 117**

**Adding the product ID to the path**

Let's start in the use folder and let's start with our product list.ejs file. There we obviously also want to be able to view the details for a given product when we click on it. Now this requires a couple of adjustments, first of all we got our entire article here which holds our product information, so this product cart and we now need a way to also load a detail page. For this I will go into my cart actions down there and add a new button there which is actually an anchor tag, I'll just give it the class button to make it look like one and there I'll add details or write details on it and this should go to some details page, we'll continue working on this part in a second. Let's now reload that products page here and you should see that details button there. Now details should of course not lead to just slash but to a page that does show us some details for the given product and this is exactly one of the use cases where we need to pass additional information as part of our path because let's say we want to load /products and then /information for this specific product, now for that, we need some unique identifier. So first of all, let's ensure that every product we create actually has a unique ID and for that, I'll go into my model product.js and in there we don't assign an ID yet, so when we create a new product here or when we save it to the database so to say, so to a file, in one of these steps we should also add an ID and I will do it in save. Right before we do save it to a file, I'll add a new property by writing this ID and this adds a new property to the entire product object we're working on and I'll set it equal to and now we need some unique ID. Well we can generate a real unique ID with some external packages or anything like this, for now I will simply take math random which theoretically is not guaranteed to be unique of course but here as a dummy value it'll do. So math random is now my unique ID, I'll convert it to a string to have a text unique ID, you could use a number but I'll go with a string and therefore this will now be saved for all new products too, so our products will now have an ID. And now that we know that, back in product list.ejs, I want to pass that unique ID to my path here. So I will now use ejs here to also output something as part of that link here, as part of this path and that will be product.ID, we have that ID available now. So this link will now lead to products and now let's say we had this random value, this would be one possible path now. Now we just want to make sure that we of course then also are able to handle that and extract that unique ID from the path in our routes file so that we or in the controller, so that we can load the correct product and show the details for it and that is the whole idea here. We send some information as part of the path so that we can extract all the data we need for the product from the controller or inside of the controller because we can't really send the entire product as part of the url but we can send this key information. Now with that, let's go to the data folder and make sure to remove all products you have in there and I'll just quickly save that link here so that I don't have to search a new image but then make sure to remove that or even easier, add an ID by adding ID between double quotation marks and give that some random number. Theoretically you can use any string but I'll go for some random number. And now we have that ID added and therefore our code will now work, if we didn't add this well we would not find an ID and therefore we could not create that path that contains the ID. If you save that and reload, if you hover over details now or if we click on it, you should see that in the url you now also have that ID in there and right now we got a page not found because we're not handling this route yet but we have products and then this ID and in a next step, we will now extract that ID and then know which product to load.

**Lecture 118**

**Extracting Dynamic Params**

So time to extract this ID from the url and for that let's go to the routes folder, there in shop.js. we want to handle a new route and I will add it below my all products route here. It will be a get route because we want to display a new page for now and part of the path is products but that's not everything, we also have this dynamic segment, the ID. The express router supports us with this, we can tell the express router that there will be some variable segment by adding a colon and then any name of our choice like Product ID, later we'll be able to extract that information by that name here so remember it. The important part is the colon here though, this signals to express that it should not look for a route like products product ID but instead that this part here can be anything and it will simply route or load this route for this path then and we will then be able to extract that information through that name. This has one important implication, if you also have another route like router get products delete let's say, so that is a normal route, this is not a dynamic segment, the order would matter. If you order it like this and keep in mind that your code is parsed from top to bottom and the request goes through that from top to bottom, if you order it like this you would never reach that route because if you had a route like products delete, expressjs would already fire at this route or would already handle it in this route here because delete would basically be treated as the dynamic segment. So if you had a dynamic segment and a specific route, you would have to put the more specific route first so that for products delete, this handles the request and thereafter it'll not continue its journey because you don't fire next but if you then have something else which does not match products delete, then you would go into that dynamic route. So this does matter, here however we don't have that case, I just want to mention it, so for now let's simply connect a controller where we then can handle this incoming request and where I then can show you how you can get this information out of the url and for this, let's go to controllers shop.js and there simply add a new controller, I will add it below get products, the position doesn't matter but logically here we get all products so now I also want to add the function where we get one product. So here I will have get product, whoops response next like this and in there let's now extract that dynamic path segment or the value it holds to be precise. So this will be our product ID, I'll store it in a constant named prod ID, that name is up to you and we can get access to it by accessing our request and then expressjs already gives us a params object on our request and on that params object, we can access our product ID and we can access product ID here because we use product ID in our route shop.js file as a name after the colon. So the name you use here after the colon is the name by which you can extract the data on this params object. And let me show you that this works by logging this, Prod ID and I will then for now simply redirect to the starting page, later we'll of course render a different view but for now this will do. And with that I can now go to shop.js and use my controller here, so connect the shop controller get product function here and with this connection set up, if I now go back and please note I'm still on that route with the dynamic segment, if I now reload this, I'm getting redirected which means we handle this, we don't get the 404 page anymore and in the console here, I see my dynamic segment logged out through this line. And that of course means we can not just log it, we can also use it.

**Lecture 119**

**Loading Product Detail Data**

Now that we're able to output our dynamic segment here, we can of course use that information to instead load that product from our product file, so through our product model which is responsible for interacting with the product and then we can return a view which does actually show us these product details. So for loading let's get into our product model now, there we got to save and we get fetch all, now I want to load a single product and for this, I'll add a new static method and I'll name it find by ID, the name is totally up to you of course but there I expect to get an ID as an argument and then also a callback which will be executed once we're done finding the product here. In this function here, I will get all my products because I basically need to read the entire file, we got no database here where I could run a query for one product only, we'll do that once we add a database of course. So for now I get all my products just as in my fetch all function here but here I now have my products available And before I return them in the callback, I want to filter out that one product with my ID. Now keep in mind products which I'm returning here is already a parsed array of products and a product will have an ID right, we assign that ID here and we store that ID. So we will have an array of objects where each object has an ID and therefore we can now use normal javascript to filter out the product we interested in. I can simply find my product and store it in this temporary variable there, I can find it by searching products with the find method, a default javascript method, this will execute a function we pass to find on every element in the array and we'll return the element for which this function we pass returns true. So this function we pass here will get the product it's currently looking at because it executes it for all the products in the array. It will pass us this product into the function and then I write an arrow function here and there is a short arrow function syntax where you can omit the curly braces if you only got one line in there and you also return the result of this one line, so there is an implicit return statement in front of that code I'll now write, so I will have one line of code here which is returned and there I will check if the ID of the product I'm currently looking at is equal to the ID I receive as an argument here and if this is true, then the product which I'm currently looking at will be returned and stored in this constant here and therefore I can then execute a callback with that product. This is a synchronous function, doesn't execute any async code, so simply having two lines after each other will do the trick here. So now we have that find by ID function in the model and now in the shop.js controller, we can use that, so instead of logging the prod ID here, we can log product, referring to our model, remember we're importing the class here, product find by ID, though we can't log this because that will be an asynchronous function, we have to pass in a callback. So here instead I pass in my prod ID and then I will get the product eventually and in that function here, I will now simply log that for the moment so that we see if that was retrieved correctly. So back on the products page, let's view the details again, that works and in the console log on the server, we indeed see the details for the product we loaded, so this seems to work. Now let's also verify that this still works if we add another product here. I'll add this and indeed we see the product here so this is working and now if I click the details link here on the products page, I also get the information for this product and the link of the image is so long here because it's not a url to an image stored on a server but actually the image encoded in base64 which is like a text encoding technique you could say. So this is the product and this is our fetch product function working. So as a next step let's add a view that displays the details and obviously feel free to pause the video or to well this video ends here but to go ahead and do this on your own, we'll do it together in the next lecture.

**Lecture 120**

**Rendering the Product Detail View**

So let's add a view or we do have the view but let's add some logic in that view and for this, I'll take the code from one of my other simple views like orders.ejs class and paste in there so that I got the head and also my navigation and so on. Now in here, I don't want to output nothing there of course instead here, I want to output my product information, so my product detail. For this I will quickly give this main element here a class of centered and find tune my css code real quick, in the main css file let's simply add a class, you can add it anywhere, I'll add it relatively at the top centered where I set text align to center for now, so that everything is centered horizontally. Now in there, I'll output my h1 tag and there I expect to get information about the product we loaded, so here I want to get some product and output the title. Now right now we're not rendering this view and we're not passing the data into it but eventually we will. So here is my title, then maybe render a horizontal line and then let's display an image for this product because we got an image, right. So I will add a div here and in this div and image and there I will output product image url, that is the field in which we are storing the image, check your model in case you're not sure and as an alt text I'll output product title. Now below the image, I want to output a h2 tag where I output my product price like this and beneath that, a paragraph with my product description, just like this. So that is my detail page there and if we now save this, we have to make sure that we render this view for our detail route, so in our controller where we do get this single product data instead of redirecting here, I'll remove that and instead of logging this to the console, I'll say res render and I want to render my view in the shop folder and there it's the product-detail view, so basically the filename here, product detail and we have to pass in some information. We do that by passing a javascript object and in there, we now need to pass in a product property because we're accessing this here in the view. So back in our shop.js file, let's pass in product and set this equal to our product we're retrieving here. So product on the right side of the colon is the product we're retrieving, product on the left side is simply the key by which we'll be able to access it in the view. With that, let's save that and go back to products and load the details here and we get an error that we're missing the page title, showing us which line is throwing the error and also tells us page title is not defined and this makes sense because in the head of our of each page, we're outputting the page title here, so we also have to pass that into our view. So in shop.js here, let's make sure we don't just pass the product but also the page title and here we can actually use product title to dynamically set the page title to the title of the product. With that simply reload this page, I'm also missing the path now because in the navigation, we're using that to determine which path is active and now there the question is which path do we want to highlight? In a navigation file here, we obviously have no path, no link to this exact product but I think it makes sense to highlight the products link here because we're still in the products area, just in the detail for a single product. If we want to highlight this, the path we should pass is /products because that's the path we're checking here. So in shop.js . file, I'll set path to /products here because this is the path for which I want to mark the navigation item as active. Now if we reload, this looks better and now here this doesn't look too shabby. We got a nice detail page here, does the trick for now, let's maybe finish it up by adding an add to cart button below the product but with that I'd say this is what we need for the moment. So let's go back to our product-detail.ejs file and below all that information, I will now add a little form which leads to cart with a post request because I want to add that product and I'll add a button with class button and type submit and the route for this is missing still where I say add to cart and now with that if we reload, we got that button here too. So this is now working and this is looking decent. With that I'd say why don't we work on that add to cart functionality as a next step.

**Lecture 121**

**Passing Data with POST Requests**

You now learned how you can display product information by getting that information out of the url, now we have a similar problem for adding a product to the cart. We get that add to cart button but in the end when posting our data, we also want to know which product to add to the cart. However here, I will send a post request right, add to cart sends a post request and this has one important implication, we can pass data in the request body. This was not possible for a get request but for post request you typically use the request body, this is also what we use when adding a product. For adding a product, we have a form and as I mentioned there when we added this the first time in the course, this automatically gives us a request which puts all the input data and so on into its body but this only works for posting data, this is not available for getting data but for posting data, we therefore don't need to add anything to the url because we can put the information in the post body. So therefore what we should do on product detail in this form is we should simply pass some important information as part of the post request. So I will add an input here and I will make it of type hidden so that it doesn't really get displayed on the page but still is encoded in the request that is getting sent and I will give this a name of product ID, the name is up to you though and now we also need to store a value here and that value will be the value that is added to the request for that product ID and here I'll use ejs again to output product ID because I still want to pass that ID to my backend so to say, to my node express app but I don't want to use the url because I do use a post request so there's no need to use it. You could do it though, you can also add a dynamic segment for post requests, that would work too but there is no need to do it here. So let's pass that data with that hidden input and let's now work on the backend on our shop.js file. There we got get cart but get cart is not the route we're working on right now, we'll use that to display the cart then but for now I need a post request. So let's add a new route which accepts a post request, still to get to /cart, that doesn't change but now we need a new controller function and we'll add one here, maybe near get cart, maybe in front of it or after it that's up to you, exports post cart and there I also will get my request response and next function as arguments. And in here we will now have to retrieve the product ID from the incoming request and then also fetch that product in our database, so in our file and add it to our cart. We got no cart yet though so this is also something we'll have to work on. For now let's simply extract the data as a first step, the prod ID by accessing request and now since it's part of the request body, we'll access request body here and then product ID because product ID is the name I'm using in my view here on the hidden input. So therefore here, I got this available, I can console log this for the moment and thereafter I can simply redirect, maybe to just /cart here and this will then load the get route, so it will render the cart page. So with that let's go back to the routes file and let's connect the newly created controller action with the post route there, so post cart should be connected here and now if you save that and you go back to the details page and for now only there this button will work and you click add to cart, I'm on the cart page and in the console we see that random ID which was generated, that random number being logged. So this is working, now obviously we're not storing that in a cart right now and before we work on that, let's actually go back to the product detail page and grab that entire form here and make sure we also use that form in all the places where we also have add to cart buttons. So let's add it on the product list.ejs file, now important there I also have product ID available, so I also access a product field so this is working here and let's add it to index.ejs and there I also have the product object available which is now important all the time of course because I'm accessing product ID here and since this is exactly equal in all three views, we can also add an include add to cart ejs and put that entire code into that include here and then simply well include that include. So instead of having that code here in index.ejs, we can use the ejs tag with the minus to add include and then we go to well up one level and then into includes and then add to cart and we use that same code also in the product detail page here and also in the product list page. So there I also want to replace this form with my include. Now let's see if that still works, if I now go to products, now we get this error because if you have an include in a for loop or in a loop in general as we have it here, we're looping through all the products and then product is a local variable available in that loop only, then in include, included in the loop unfortunately doesn't get that variable by default but you can pass it to that include here and you can do so by adding a second argument to the include function where you again pass an object and simply add that variable again. So here this product is what will be available in include and on the right side, you include or you add the value you have available in this file, so product in the loop is now passed to product in the include. This of course needs to be done for all our include, so also in the index.ejs file and not for the one in the product detail though because there, it's not inside a loop, so it's available globally and therefore works and now if we reload here, this works, starting page works and the details page also still works and we can go to the cart on all these pages. This is looking good, we're also outputting the ID here so this is working. Now as a next step, we'll have to work on that cart.

**Lecture 122**

**Adding a Cart Model**

We're now working on the cart and right now, we don't have a cart, we got a product but not a cart, let's now add a new model here, the cart.js file because the cart is like a separate entity in our project you could say. Therefore I'll again export a class here and I'll name that class cart, so that is pretty similar to what I do in product, there I export the class product but now it's class cart and now we have to think about how we want to manage that cart. Now obviously we want to have a cart that holds all the products that we added and we also want to group products by id and increase their quantity in case we add a product more than once. So to do all of that, I will first of all create a constructor here which allows us to create a new cart. Now when this cart is created, I'll add a products property here which should be an array and there I imagine having some objects in there which do maybe hold information like the ID and important, also the quantity of this product. We can also add an information like total price which initially is zero let's say and this of course increase with every product we add. Now what we need on this cart though is a way to add and remove our products obviously. Now the problem we have is the cart itself is not really an object we'll constantly recreate, not for every new product that we add we want to have a new cart, instead there always will be a cart in our application and we just want to manage the products in there. So the approach I want to take will actually be a different one, I don't really add a constructor instead I'll add a static method, add product like this. Now this will take the ID of the product I want to add and the goal here will be to then fetch the old or previous cart from our file for now, analyze that and see if we already have that product, find existing product and then add new product or increase the quantity. That is what I plan to do, so let's start with adding the logic for fetching a cart from a file. For this I'll import the file system here and also the path helper here, whoops, path to construct a good path. Now in product.js, we see how that path should be constructed so we can copy that basically, go to products, to cart.js and then add it here. Now the difference is that the file will now be named cart.json and in there we'll store an object that represents our cart and then here, in this part here for adding a product, I want to use the file system to read a file and that will be the file at this path so my cart.json file and we have a callback where I either get an error or or the file content. Now if we have an error, we know that the file doesn't exist yet and therefore we got no cart yet. So if we got an error then our cart will have to be created otherwise we know that we, well get an existing cart. So here I'll add a new cart first of all which will have products that are an empty array and maybe that total quantity we were talking about, the total price excuse me which is zero. And therefore if we don't have an error, again that inverse logic, if we don't have an error then we know we got an existing cart, so in this case my cart should be equal to the parsed file content. We'll store it as json so I'll parse that with that json helper. So now at this point after the if statement, we know that we will have a cart and now we can analyze it and add a product, so let's analyze the cart and see if the product we're trying to add already exists. So we'll search for the existing product by taking our cart products in there, remember we'll have products in the cart which is an array and by then finding an element in there. So again we'll loop through all the products and have a look at each of them, each prod, so in each product we'll have a look and we'll see if the product ID matches the ID of the product we try to add. Now if we got an existing product already, then we simply want to increase that quantity. So let's say we assume that each product object that gets stored in there is not just a product object having the data in our product model but also that it has an extra quantity field. Now if we have an existing product, then I want to create a new product and for this I'll create a new variable, updated product and here in this if statement, so if we have an existing product, I'll use updated product and now next generation javascript with the object spread operator, I'll take all the properties of the existing product and add them to a new javascript object and then on that updated product, I'll set the quantity equal to the old quantity plus one. So I'll simply increment the quantity by one and since I distributed all properties of existing product into updated product, the quantity will be available there already because it was available in existing product. Now if we have a new product, I'll set updated product equal to and I should put that into an else block, so if we have a new one, I will set updated product equal to a new javascript object where I add information for that product and that will be my ID let's say. So ID will be equal to the ID I'm getting as an argument and I'll set the quantity to one because we added just one. Now the remaining thing is that I want to update the price of the cart. Now we got the cart here with the total price and the total price will always rise by the price of the product we added. Now I don't have that information in here though, so I expect to get it as an argument, product price and therefore here after this if else block, I can say cart total price equals cart total price plus product price right because we want to, price because we want increase that. So now we got our product added, we found it and we also analyzed it and added a new product, increase the quantity, now we just need to save the cart back to our file. And of course the cart should now also contain the updated product, that's also important. So we updated the price thus far, let's now also update the products, so the products on our cart should equal the old products and here again I'll use a next gen javascript feature by spreading the existing products of the cart, so this will now be an array with all the old cart products and now the question is what do I want to do? Well if we are creating a product for the first time, so if I'm in this else block, then I simply will add the updated product as a new additional product. However if I got an existing product here, I don't want to add a new product instead I want to replace the old one and to do that, I need to find out where in my old products this existing product was located, so which position it had. To do this I'll get the index instead of the product, so here I'll have the existing product index and then I'll add my existing product which is simply cart products at this existing product index, so one extra step but this now allows me to use that index to replace the item in our cart products here. So there, I will set cart products equal to cart product, so first of all copying the old array just as we're doing it down there but then I'll not add updated products, instead I will set cart products and overwrite existing product index, so at this position I will replace the element with my updated product. So now the updated product is either replaced or added to the cart products and the price is updated, now we can save it back and for this, we can use the file system write file and write it to that path and then of course also define what we want to put there and I want to write my cart into the path, obviously in a stringified version, so as json and then here I have a callback where I might get an error which I want to output so that I see if I do have one and which one it is if I got one and then I am done. So this is now an add product method that hopefully does the trick, let's simply try it out. So for this, let's go to the shop controller and in post cart, I want to add my product. This of course means here first of all I need to get the product because I need its price too, so I will use product find by ID for my product ID and then I have this callback where I get my product, so here that is the product that is retrieved from the products database so to say, from the products file and once I have this, I can use the product information to update my cart. So in here, I now want to use my cart model. First of all, let's import it, cart by requiring it from the models folder and there from the cart folder and with it being imported, let's go down to post cart and say cart add product and now cart the model basically serves as a utility model you could say, we're not instantiating it instead were using this static function and I'll use add product to pass in my prod ID and also my, here product that is what I'm retrieving from the product file, my product price because that is also information that I need in there. With that let's see if that works by going back to our app here and clicking add to cart, now looks good because we're logging the error here and if we see null that means there was no error and now we got cart.json and in there, I got products which is an array of products with an ID and the quantity and the price is stored as a string which is a bit suboptimal here I have to say, so we'll have to work on that but besides that, this is looking good. Now if I add another product or the same product again, then we see indeed the quantity was increased, the price is misbehaving because it's stored as a string and therefore concatenated, so we'll have to do something about that. In cart.js the price we're extracting, it's stored as a string in our product model, so what we have to do is here when we work on the price, I have to add a plus in front of product price to convert that string to a number and now if we quickly delete cart.json to start from scratch and I add my element to the cart again and I do that again and we now look into cart.json, now this is looking better. So this is now working, this is our cart model added but with that let's go back to our routing topic which we had in this module and let's see what query parameters are and how they can help us with editing a product.

**Lecture 123**

**Using Query Params**

We worked a bit on the cart model, some things are still missing for example we never display our cart and we also never really, we're not able to delete an item from the cart but let's ignore that for the moment, let's instead focus on editing a product. We've got the edit product ejs file and in the end what I want to do in there is I want to render the same form I have for add product, the difference will be that I want to pre-populate that form with the values of the product I want to edit. Now if I'm using the same html code in the end, it would make sense to reuse the same template. If you ever plan on using a different one, it doesn't but here it makes sense. So what I will do is I'll take my add product.ejs code and move it into edit product because I find this to be the more generic name and I'll delete the add product.ejs file, so now we just have edit producte.ejs with all the html code in there that we need. Now first of all, that means that I have to load that file in admin.js when we load get add product, I no longer need the add product path here, I need edit product here only, the part down there can stay add product because that is what we use for highlighting a certain navigation item. By the way we can get rid of these additional information pieces. So now this should still work, if I click on add product I still see that form, so that's looking good. However for editing a product, I now also want to have a route and I want to use that same view, so I'll copy that route here and add it maybe below post add product to keep these two together and here I will have get edit product as an action name and I also want to render edit product here. The difference is that here I plan on passing in my product information but the problem is of course how should I reach this controller action and my idea is that since I already have my edit button here, if I click that I don't want to call edit product like this, I also want to add the ID of the product I want to edit. So if I grab a real ID from my products.json file like this one, then I would have that in the url and if I load this, now I want to have the form pre-populated with the data for this product and if I hit the save button, I of course don't want to create this product but simply edit the existing one. Now this means that I need two things, the ID I need to pass that and the information that I want to edit a product instead of create it. Well if we go to our admin routes, we can add a new route first of all, this edit product route here with the ID and you learned how that works actually, you can add a new route here, get route will be at slash and then here, the /admin is available for all routes in the admin.js file anyways as you know, so it's just edit product and then the ID and again this is a variable, a dynamic path segment indicated with a colon. And then we can load the admin controller and there the get edit product action we just added. So this is step number one and actually, we now already can go to our admin.js controller and here we obviously know that we want to edit a product. So what we can do is we can pass an additional information field to our view editing and set this to true maybe so that we can check this with an if condition to find out for example if upon clicking that save button, we should try to add the product and send a request to that route or try to edit it and send it to a different route. But let's say we want to get an additional confirmation by ensuring that people have to pass us a so-called query parameter in the url. A query parameter can be added to any url by adding a question mark and then a key value pair separated by an equal sign, like edit equals true and you can have multiple query parameters by separating and them with and percents, so we could also have title equals new for example. So this is possible and this is so-called optional data, the path here, the route which gets reached is determined by the part up to the question mark. So you don't need to add any information about query parameters you might get to your routes file, these paths are not affected but you can

**Lecture 124**

**Prepopulating the edit product page with data**

So to pre-populate this form with the product data, we need to fetch the product first, so for in added mode here after not being redirected, then I want to get my product information. For this I need the product model which I already have and I need the product ID, prod ID can be retrieved from the incoming request because if you check the routes real quick, we have that dynamic segment here, so by this name we can extract the Product ID. So in admin.js, here I can set or I can access request params product ID and get that ID from the URL. Therefore I can then use my product model and find this product by id and then here, I have my callback where I receive the product that was retrieved and by the way of course, you should also try to add a check to see if you have a product and redirect the user in case you don't but let's do it like this for now. So here I will now get my product and I will then render this page assuming that I always get a product and pass my product on a product key and you can name this key whatever you want of course. And we can for example add a check if we don't have a product, so if this is invalid, if it's undefined then we could return a redirect for now which is not the best user experience, most of the time you would want to show an error instead but let's now, for now do it like this. So we assume we always make it here and we get the product in the view now, so let's move over to the edit product view. And here first of all let's work on that button. Right now it's add product but I want to change that button caption if we're in edit mode. Now remember we're setting this editing property here so this is a variable available in the template. So here I simply want to check if editing and then add a curly brace, if this is true then I actually want to display the edit or maybe update product text here on the button otherwise and for that, I'll close the first if block here and add an else block still in my ejs tags of course. So otherwise here if we're in this else block here, I want to show add product and thereafter I also have to close my curly brace of that if statement. If I now reload this page, we see update product correctly and by the way if I go to add product, I'll now get an error because editing is not defined. So to fix this, I also have to make sure that in my controller, for showing the add product page, I set editing to false of course. And now here reloading this page will work, going back to the editing page will show us the update button. Now that is one step, back to the template I don't just want to show the caption here, I also want to change the action. So here on the form, the route we're sending the request to should also change of course. Right now it's always add product but it should only be add product if we are not in editing mode. So here indeed I will also output something dynamic, I will check if we're editing, so I'll have the same logic as for the button and if this is the case, I went to load /edit product let's say, otherwise I'll close that and have my else block open up, so otherwise this is my ejs else block, I'll add add product into my url here and I can inject these segments into the url because this will just be converted to normal text in the end. This is what ejs does or or what all templating engines do. So also close that curly brace of the else block here and with that if I save this, we should make sure that if we click that update button, we go to edit product. Well let's go back and let's see if we can pre-populate this with product information. Now keep in mind that in admin.js, I am retrieving the product, whoops not here but here in get edit product and I pass the product information into my view. So therefore in the view, we can of course use that and we can set it on all our inputs, here on value, I also want to check if I'm in editing mode and if I am, I want to display my product information, if I'm not I don't want to do that. So same logic as before, I'll check if I'm editing, if I am then I will display product title here because this is the title input otherwise I will not do anything and therefore I don't even need an else block, I just do this if I am in editing mode otherwise nothing will happen. So if I now reload this, we indeed see product title and the issue here is that of course this is simply output as text, to output the real value behind it, we need that equals ejs tag here nested into our other ejs tags like this, so this was now added, equals unclosing with a dollar sign greater than sign. And this can be hard to read, you simply have to divide it up in blocks, you've got if, then you've got the output and then you've got the closing curly brace for and with that saved if you now reload, you'll see the real title behind that. Now let's do the same for all the other values, so I'll just copy that value here real quick, for the image url. I obviously want to output image url, for the price. I want to output product price. For the description we get a text area, so here I will not have value equals, I'll just output the value between the opening and closing text area tags and here I'll output the description. And with that all saved if we reload, this is looking good and now we can start editing our products. Now I want to hook up that edit button so that we can really load the product and this newly added view. So let's do that and then also work on the functionality to store the updated information over the next lectures.

**Lecture 125**

**Linking to the Edit page**

Let's make sure we can reach that edit product page through a route and for that on our products page here, on the admin page this added link here, added product should also include the ID of the product and again we can use ejs to simply inject that into our path by accessing Product ID. That's all we need here, if we now reload this page and click edit, we get redirected, what could be the issue here? Well remember that we added our query parameter right and we're checking this query parameter in our controller and if we don't find the added query parameter, added mode will be undefined and therefore we get redirected. Now this is of course just a bonus but it's important for you to understand how query parameters work and therefore we have to set this too when creating our link to this page. So let's go back to products.ejs and don't just add this ID but also simply append that query parameter where I set added equal to true. With that if I now save this and I go back to admin products and click edit, now we load this product for editing. So this is looking good, let's now make sure that clicking update product which does already reach the correct route does then also do something. And for this first of all in admin js, we need to register this route. So let's add a new post route which is at added product, this will not receive any dynamic segment because it's a post request so data can be enclosed in the request we're sending, so let's now work on the controller here and add a new action, post edit product which of course receives the request object, the response object and the next function as all our middleware functions which our controllers just are and in here, what do we have to do? Well we basically want to construct a new product and replace the existing one with this product This means that we have to do some work on the product model, we'll do that in the next lecture.

**Lecture 126**

**Editing the product page**

Time to work on our product model again. In there we get a save method and right now we used that to create a new product. Now why don't we also use that for updating an existing product if we already have it? To do this what we'll have to do is in save we just have to check if we already have an ID and therefore when creating a new product, we should accept an ID too and then set this ID equal to ID but we'll simply pass null here for a brand new product, so that we can still create products that don't have an ID yet then the ID will be assigned here but if I we're editing one, we do have the ID already so we can simply assign it here. And then in save, we can simply check if this ID is already existing, if it is null, this will fail and will automatically make it to the next line which we want but if we do have an ID, save should not create a new ID and new product, instead it should simply update the existing one. We'll still have to get all the products though, so indeed this should be moved into our callback because we need all the products anyways and the new ID creation can also move in there but after this if statement. Now in this if statement, I now want to update the existing product and for that, I need to find it first. So I'll find my existing product index again by searching for or going through all my products with the find index method, products will be an array as we know and there, I will get access to all my products stored in the temporary prod argument here or in the prod argument of this anonymous function I should say and I can simply check if the ID of the product I'm looking at in this array is equal to this ID, put in other words if I'm now looking at the product I plan on editing. If that's the case then I found the index of the product I want to edit and now I simply have to replace that in that products array. So I'll create an updated products array where I use that spread operator again to pull out all the existing product elements, store them in a new array and then on that array updated products, I'll replace my existing product index with this because this inside of this class here is of course the updated product because you have to imagine that I create a new product instance, I will populate it with information about my existing product and then I just call save and I will find out that I already have this product and therefore I just replace it in the array which is stored in the file with the newly created product I'm in. So with that being saved, I just have to write that information to the file, so fs write file is what I need to execute, so this code will stay the same, just that I need to call it on updated products here and I will now also wrap the other part here in the else block so that not both snippets execute but only one of them. So now we're storing the updated products and write file will always replace all the old content, so we won't add it or anything like that, it will replace it and therefore we should now have a save function that we can use both for adding new products or editing existing products. Now this has one important implication, we now need to go to the controller and when adding a new product here in post add product, we now also need to set null as an ID, as a first argument here on our product constructor because we just added this as an additional argument here in the constructor and if it is null, then this check will fail and we will therefore make it into the new product created mode which is what we want. We now can also work on the post add, edit product method, there I need to do two things. First of all I need to fetch information for the product, then I need to create a new product instance and populate it with that information and then I need to call save. Let's first of all extract the product ID, prod ID by accessing the request and there since it's a post request, I expect to get that information in the request body. However at the moment this will not happen, so let's go to the view first to the edit product.ejs file and there I need to add a new hidden input which stores the existing product ID. However that is only an option if I'm editing a product, not if I'm adding one, so first of all I'll use ejs to again check if I am editing and also close that here, here we also need to open a curly brace therefore and if I am editing and only in this case, I'll render a new input here which is hidden, so which the user can't see where the value is now my product ID, so I'm using ejs to output the product ID there. And this will now therefore be included in the form, now I just need to give it a name, product ID maybe and now I can extract it by that name in the incoming request in my controller, so request body product ID because I used product ID as a name here in the view on the hidden input. Now with that ID fetched, I could fetch my product through the product model but actually this is the edit route right, so I get the new values I want to store as part of my post request body because the user enters them here in the form. Here all this is sent to me, so I will now simply store all that in values or in constants like my updated title will be request body title, I'll have my updated price which is request body price, I'll have my updated image url and you can name these constants however you want. Now important of course is what you access here on the request body, these keys have to match the names you have on your inputs in your well added product view. Last but not least, we got the updated description here which is request body description. And now with all that data, I can create an updated product, this name is also up to you, instantiate a new product therefore and here, I do pass my existing prod ID as the first argument and this will ensure that in the product model, in this check here we do find a valid ID and therefore we go into this updating mode instead of the add mode. So I'll pass that ID and I'll pass my updated title then, I'll then also pass my updated image url, I'll pass my updated description and my updated price. So this is what I pass and now thanks to our changes to the product model, I can call updated product, save and it should hopefully just save that and override the existing one. Now let's try this out and for that, let's go to our admin.js routes first of all and register this newly added post added product action on the added product post route and now let's reload our page here and add a couple of exclamation marks and hit update product and the problem is I never send a response so this failed but if we have a look at our products.json file, the first product didn't change but the second one indeed has all the exclamation marks, so this does work. So the missing thing just is that we send the response in our controller, so here after calling save, I will actually call res redirect and go back to just /admin or let's have a quick look at the admin.js routes, I want to go to /products, so /admin/products I mean, this is what I want to redirect to. So let's save this, go back and reload, here we see the estimation marks too, let's change the price to 30.95 and also change the description, it's really great. Update the product, we are redirected and we can already see the changes here and we can edit it again of course, for example remove the estimation marks. So this is working, we're now able to edit the product.

**Lecture 127**

**Adding the product-delete functionality**

Now the application is taking shape, now there are two things I want to add in this module still before we're done. The first one is I want to make sure that we can also delete products and I want to make sure that we also can see our cart items on the cart view and also remove them from the cart. Let's work on deleting products here first and definitely feel free to try this on your own, you'll have to add a new method somewhere and basically make sure that you can not just save products in the product model but also delete them but of course we'll then also do it together after a short pause that you can use to pause the vIDeo and go ahead on your own. Were you successful? Well let's see how I would solve this. We want to be able to delete our products right and if we go to the products.ejs file, this view in the admin folder, we already have that delete button which is wrapped in a form on purpose to send a post request to delete product, so we need to handle that route, that is a good starting point. So in the admin.js file in the routes folder, we need a new route for this, post route for delete product. Now since it's a post request, we also don't need to enclose or encode any information in our path in the url, we can put it as part of our request body instead. So back in the view, here I will add a hidden input again as we did it before in post requests where I set the value to product ID using ejs templating syntax and the name to product ID so that we can extract that information by that name. So that's the first step, now we do have that route, obviously we also need to work on the controller now. So let's go to the controller, the admin controller and let's add a new action for deleting a product, exports post delete product. There I get my normal arguments in this function and now the question is what do I want to do in there? Well obviously I can extract the product ID from the request body by accessing product ID, that is why we just added it to our ejs template and now we can also go back to the routes file and simply connect this route to our admin controller post delete product action but the real magic now happens in our model, in the product model. We have the save method, now I want to delete a product. So let's add a delete method in there and what I plan on doing here is I actually want to turn this into a static method and maybe name this delete by ID, pass in an ID and then have all the logic for deleting a product in here. To do this I first of all need to find out which product to remove or update my array of products so that I can then write it back to my file. So just as in find by ID, I will call it get products from file to get all the products and in here, I then have the products array, there I now need to find the index on the product I want to delete. So I will use products and then the find method to go through all my products and find the product with the ID I'm trying to delete, this is a mechanism we did before, whoops, it should be find index though. Now this means I can update my products array such that this element is removed and actually there is even a shortcut we can use. I can create a new constant, updated products, take my existing products and not use find index but use the filter method instead. Filter also takes an anonymous function and will return me all elements as part of a new array that do match the criteria my function returns. So if this returns true, the element is kept. Now I want to keep all elements where the ID of the element is not equal to the ID I'm trying to delete because all elements where the ID is not equal should be kept around, should be part of the new array which will be the array I save back to my file. This means that in this function, I actually want to return the opposite here. So only if the ID is not equal to the ID I'm looking for, I want to keep the item so this will then return true if the IDs are not equal therefore the item is kept and only for that single product I'm looking for this will be false, so the item is not kept in the new array. And that simply means that I can now go ahead and save my updated products which are all products except for the one I want to delete back into the file. So I will use the file system to write a file on this path and simply add my updated products in json format and then again I can use my or add my callback function to see if this throws any error. Now if it does not throw an error, so if everything went fine then I also want to remove that product from the cart of course because I can't have it in a cart if the product doesn't exist anymore, right. So as a next step, we'll also work on the cart and make sure we can delete items from there, a functionality we need anyways.

**Lecture 128**

**Deleting cart items**

So let's edit the cart model now, we got the static add product method, now I need a new static method which is delete product. There I get an ID of the product I want to delete and I also want to get the price of that product, you can also name this product price because I'll need to update the total cart price of course. So with that, what do I need to do? Well first of all I need to get my cart, so I need to read my file here and copy that line and of course you could also refactor this for now I'll simply duplicate the code here to make it a bit easier to understand. So there I read my file, I try to read in the cart and if I got an error here, I can already return because that simply means somehow I didn't find a cart, so there certainly is nothing to delete right so I can just ignore this. Otherwise we will continue so if we get no error, we'll continue and now is the part where I want to update the cart. For this, I'll have my updated cart here and take all the properties of the old cart and put them into a new object with that spread operator, I now want to update both the products and the total price. The cart total price should be reduced by the product price, however this would be incorrect because if we have the product in the cart three times, it should be reduced by the product price times three. So let's postpone this for later, let's first of all find out how often we do have the product in the cart. And for this I'll have my product index and I'll find the product in the updated cart in the products array with the find index method and there, this goes through all the products and I'll check for the product where the ID matches the ID of the product I tried to remove from the cart. This is now my index of the product, however here I could even find the product right away, just product like this because I will remove it differently from our products array. For now I just need the product to find out what the quantity is because remember, we're storing the quantity in that qty field, so const product qty equals product qty, you don't need to serve that in a separate constant, just a bit easier to read and now with that quantity, I can use that here to have the cart total price be my cart total price minus the product price I'm getting as an argument times product, whoops, product quantity. So now I'll reduce the total price by the product price times how many times I had the product in the cart. So that is one important piece of information and now I can also update my cart products here, whoops, products and set it equal to, whoops not that cart by the way, the updated cart down there, so updated cart total price, updated cart total price and updated cart products is now updated cart products which for now is the old products array but now I'll call filter again just as I did in the product model, there this function runs over all elements in there and keeps only ones where I return true, so I want to return true for all products except for the one I'm removing, so here I will return true only if the product ID is not equal to the ID I'm looking for. So all other products are kept. And with that, I can write the cart back into my file by copying this, write file and store my updated cart there. So this should now in theory work, I can now also go to the product model and there I now need to import my cart model, so const cart equals require cart like this and now with that imported, in here I can simply call cart delete product and pass the ID and I also need the price now. To get this, I will extract my product before removing it so I'll have my product here by searching my products with the find method for the product where the ID matches the ID I try to delete, so now I got that product information here which allows me to pass in the product price to my delete product method. So now we are able hopefully to delete a product by its ID and also to then delete it from the cart if it was in there and right now for example, if we have a look at our cart, we do have that ID, the milk can, we do have that in there so this should be gone if we delete it. So now in the admin controller in post delete product, we can use product and then delete by id and pass in our prod ID here, like this and then we can also redirect to /admin/products. Now by the way a little side note, it would be best if we have a callback in delete by ID so that we only redirect once we're done and the same is also true for updating by the way. There the callback in save would be best for redirecting, for now I'll not do this, later we will add this little functionality. But let's now see if that works, if I delete the milk can here, I get an error, cart is not defined in my cart.js model, deleting the product work though, if you check products.json, you don't find it anymore so this worked but in the cart model, yeah I'm trying to get all cart fields but of course I'm looking at the file content so I should use this. This is a bit annoying because now I deleted the product whilst I still have it in the cart but it's no problem, I will simply recreate it and then hack the ID into the products.json file. So let's go to admin products, add a new product real quick, test, get that image again, set the same price so that I correctly update my my cart price here and remove the correct price and then some description test. So now this exists again. Now let's just make sure that I copy the ID that I have in a cart and replace the one that was newly generated in products.json so that we really well delete the correct one from the cart too. Save all of that and now let's give this a try, if I click delete, if I reload this first of all so that we load the new ID here too and I click delete, cannot read property, find of undefined is now the problem in the cart model. And that problem here is that of course is still a string json format so I need to parse it before using my file content. So let's save this, same problem as before, I will quickly re-add this, add some dummy text as an image, doesn't really matter, set the price, some dummy text here, wI will not see the image here because it is not a real link but that does not matter. Now before doing anything, let's go to products.json and again replace the newly created ID with the ID in the cart, like this, reload this admin products page and click delete and now this looks much better, now this seems to work and in the cart indeed, products is empty and the total price was reduced to zero. So this is working and now let's finish this module up by also making sure that we do display cart items on the cart view here.

**Lecture 129**

**Displaying cart items on the cart page**

We're already able to work on the cart, now I want to display cart items on cart.ejs. There I simply will add a main section before the end ejs include, that's important and now first of all, what do we need to do here? We need to check if we do have products and if we do have them, then we want to display them and I will simply display them in a really boring list for now, we can add styling later, just want to see if that works. So let's go real quick to the shop controller where we load the cart and there first of all we need to be prepared to fetch all products because we want to return them to the cart. However of course not all products we have, just the products we have in the cart. So for this we need to go to the cart model first of all and get delete product and add product, now we need a way to get all the products in a cart. So I'll add a new static function, get products here and in this function I will access my file and simply return the product IDs. I need to get a callback therefore which I can call once I got the products and then I can use this function here to read the file and in here I know I get my cart, this time doing it correctly with json parse file content, this is now the cart and actually I'll name this get cart and not get products because I will return the entire cart here in my callback, this is all I want to do. Now obviously this can fail if we have no carts yet so I will check if we have an error and if that is the case, I will call callback null here and otherwise in the else block here, I will return a cart with valid values. Now we can go to the shop.js controller in get cart and here I call cart get cart. I will have to pass a function there, the callback function I just added in the cart model where I will eventually receive the cart and I will render my view inside of this function. However I don't just need the cart, I need a little bit more information about the products too, so I will use my product model then to fetch all products here and I will add my callback here and make sure that this is nested in the cart function and we're having a lot of callbacks here but it's still readable, later we'll also find another way of working with a lot of depending async actions. So now I got my cart and I got the products, now I just need to filter out the products which are actually in the cart. So I will go through all my products, so for product of products looping through all the products I have in there and I will check if this given product is also stored in a cart and I can check this because in my cart, we got the product ID, right if you have a look at cart.json here, we'll store the product IDs. So I'll check if carts products, if I do find this product here, so if the product I'm looking at in the cart, if the ID of that product equals my product ID here and of course this is some code that can be improved if you store large quantities of data but we won't store large quantities of data in our file here anyways, we'll use a database for that and therefore for now this is perfectly fine. And if I do find an ID in the cart, then I know well the product is part of that and therefore I will create a new constant here, cart products which is an array and I will simply add that product to cart products then, so cart products push and I will add the product I'm currently looking at in this iteration of this loop. So this product I'm looking at, if it's part of the cart I'll add it to cart products. This means that after the loop is done, I'll have an array of cart products that contains all the products which are indeed part of the cart. Now important, just adding the product like this does not suffice instead I'll add an object with a product data field which holds the product and also a quantity field because the quantity is stored in a cart too but not in that product I'm looping through because these are just the products I'm storing in the product.json file, obviously there is no quantity in there. In the cart however, every product I store there does have a quantity attached to it. So in shop.js here, I will also store the quantity of the product I'm looking at. Now I'm exracting my cart product here, so actually I'll store this const, cart product data, I'll store the cart product I'm finding in the cart products array in cart product data and I'll check the existence of that then and if that exists I know there is some data about this product in the cart, I then store the original complete product data from the product model and from the cart product data, I'll take the quantity. So now we got quantity and product data in there and we push that object to cart products and cart products is now what I want to return, so here I'll return a products field which holds cart products, this is sent to my view. Now if we have no products in the cart, then cart products will simply be an empty array and I can check for that in my template. So now let's go to the cart.ejs file and first of all I'll add an if statement and check if products and now remember, products is simply this key here I'm sending into my view, so I'll check if products length is greater than zero and then close that and I'll also already define an else block here because I want to render something if we got no products in a cart too. So here in the else block and you need to close that curly brace thereafter, in the else block I'll render h1 tag where I say no products in cart, like this but here of course I want to render the products in the first block of the if statement. So here I'll render unordered list for now where I simply loop through all the products, again using ejs products for each product and that is an anonymous function here like that, then close the forEach loop here by closing the curly brace of that anonymous function and then the bracket of this forEach method here. Now in here I want to output the product information and now it's important to keep in mind that product here, maybe we should name it product data, that this does actually hold or does actually refer to a product with a product data field and a quantity field and the product data field is the product I'm interested in but I also need the quantity. So maybe even I don't name that product data but just p and then in here, I can output p.product data, referring to that nested product data which holds the product information like the title, so .product data.title would allow me to output the title. And then also maybe in brackets and this is no ejs syntax, this is normal hardcoded text here but then in there I use ejs again to access p.quantity and remember p simply is that object we passed t that cart products array on our backend code. So now let's see if that works, if I save this and I reload this cart page, we see no products in cart. If I go to products here and I click add to cart, we indeed see that on our page. If I add this one more time, we see it twice. So this is now working, the last step is that I want to be able to delete it.

**Lecture 130**

**Deleting Cart items**

Let's make sure we can also delete items in the cart and for this in cart.ejs, I will change my list item a bit, in there I'll add a div first of all or maybe let's add a paragraph to be semantically more correct which will output the text for this cart items, so things like the name and the quantity and then I will also add a little button there, wrapped in a form. The form should go to cart delete item maybe, the method will be a post method and in the form I will of course have my button with a class of button with a type of submit which says delete. If we now reload the cart here, we see delete this is the button, we can certainly work on the styling but for now I just want to make sure that it works. Now this is the delete button and if I click that, it well should delete this item of course. So cart delete item is a route we now need to add and therefore let's go to routes here in shop.js because of course this is a customer action, our customers will manage their cart and there below the cart I'll add a router post method, cart delete item. Now we need a fitting action in the controller, so in a shop controller here and there I got get cart and post cart, now I will also have exports post cart delete product maybe, you can of course use any name you want and in here we'll have to remove that product from the carts but only from the cart, not the product itself. Well to do that, let's first of all extract the product ID from the request body product ID and therefore we need to make sure that we pass it there too, so in our form let's again add this hidden input with a value of our p product data.ID and then a name of product ID so that we can extract it by that name on our backend. And with that added, with this hidden input added here, let's now go back into the controller and with that ID let's access the cart and then there, we can delete a product right. Now that delete product function takes the ID of the product and we do have that but it also takes the price and therefore, we should get that product information first. So let's also call product find by ID here and get that product for this ID and add a callback with the retrieved product, simply so that we can get the price before we well issue the delete request. So in there, in this callback I will call delete product and now I can pass in product price too and of course we could have also used a hidden input to pass the price to the backend but I think this is the cleaner approach, if we only pass the ID through the request and then we do all the data retrieval on the backend in our node express code. So now we get delete product and with that we can also send a redirect request back to cart and in theory, this should be all we need. Now of course the missing thing is that we connect our cart delete item route to the newly created shop controller action, the post cart delete product action. And with that added in the shop.js file in the routes folder, we can reload our cart page here, click delete and we see our products and by the way you can ignore these errors on the right, these are related to my local network here. So now I got no products in the carts and if we have a look at our cart.json it's indeed empty, product.json still has the products though. And if I quickly add another product with no real image, like this so that I have more than one and I add both to my cart and I delete just a book here, only that is removed, also in carts.json. So this seems to work just fine, let's also delete this one now, seems to work good and with that, we improved our app a lot and you hopefully get a bit of a feeling for how you can pass data around and how you can work with your models. Now correctly you would say that of course our current approach here in the modules is a bit suboptimal because working with the file and so on, it's not that great, we can also improve some things in the controllers because for example as I said, we should redirect if we know that the deletion succeeded and not right after this line because since we access a file in there, theoretically we should use a callback here too. But these are all things I will do once we finally also added the database, something we will do in the next module now. So time to move on to this exciting big block of the course.

**Lecture 131**

**Fixing a delete product bug**

Now I'm generally pretty happy with the state of this application but there is a minor issue in it. If we have no products in a cart or at least not the product which we're trying to delete, let's say the one on the right here, we actually get an error and we do get an error because we try to access the quantity of a product which we just don't have. And the reason for that is that we simply use in our products.js file in the models folder, we use the cart model to delete the product from there, right. The problem just is obviously not every product is in the cart, so in the cart here where we have delete product, we first of all need to check if a given product is part of the cart. So here when I parse the overall cart, here when I find the product in the cart, I should check if we really have that product because if we don't have it, so if I add an exclamation mark in front of this, so if this check here is true, that means we don't have a product then I simply need to return here, I don't want to continue, I don't want to try to edit it because it's not part of the cart. So therefore now if I reload my app and I quickly add a new product here, whoops, with a price and a description like this and I then click on admin products and delete this, now this works.