**Lecture 96**

**What is the MVC?**

So what does mvc stand for or what is it? It's all about a separation of concerns, so making sure that different parts of your code do different things and you clearly know which part is responsible for what. MVC stands for model view controller so we work with models, views and controllers and actually for example views, that is something you already know, we already got views in our project, right. Models are basically objects or is a part of your code that is responsible for representing your data in your code and allowing you to work with your data, so things like saving data, fetching data to or from a file or even if it's just in memory as we're currently doing it, this should be handled by models. The views are responsible for what the user sees in the end, they are responsible for rendering the right content in our html documents and sending that back to the user, so they are decoupled from your application code and are just having some light or minor integrations regarding the data we inject into our templating engine to generate these views. And the controllers are now the connection point between the models and your views because since the views shouldn't care about the application logic and the models do care about how to save and fetch data and so on, the controllers are the thing working with the models, saving that data or triggering that save process and so on and also the part where they then pass that data that was fetched to your views for example. So the controller is the middleman, it contains the in-between logic. Now in case you're also wondering how routes fit into this picture, well routes are basically the things which define upon which path for which http method which controller code should execute and as I just said, the controller is then the thing defining with which model to work and which view to render. This is the pattern and in an app with express or built with express as we are doing it which heavily relies on this middleware concept, the controllers are also kind of split up across middleware functions or some of the logic might be separated and moved into another middleware function but we'll see all of that and we'll get there step by step. For now, let's simply move back to our project and implement an mvc pattern there.

**Lecture 97**

**Adding Controllers**

In our project as I mentioned in the previous lectures, we already got some views and we'll leave that exactly in the state it is. We've got that views folder with all our templates inside of them. Now what's missing is a folder for the controllers and the controllers themselves and for the models. Right now that is all mixed into our route files or into our route functions here, the way we route won't change, we use the router and we have such a middleware function but actually, the logic that is executed here that is the typical controller logic, we're interacting with our data even though that's just one line here but still and we're then returning a view and that's exactly this in-between logic that makes up a controller. So therefore you could of course say well we already got controllers, these two files hold our controller logic and you would be right but especially as our application grows if you put everything into your route files, this can quickly become a very big file and therefore separating this into separate files can be a good idea because you can then quickly see which routes you have and if you want to see the code which executes per route, you simply go into the respective controller file and function. So therefore let's create a new folder here and let's name it controllers, like this. Now in that controllers folder, I will create controllers for the functionalities I have. Now you can have a one to one mapping between your route file names and route file number and your controller files but you can also split this differently, maybe you want to group your routes by prefix, like here, these are all our admin routes but the thing you execute there might fit two different controllers. Let's say you have admin routes which allow you to change your admin user profile and products, you might have all these routes in the admin.js file in the routes folder but you do have two different controllers, the users controller or admin controller and the product controller. So this is up to you and I will indeed create different files here, I'll have my product.js file or products.js for my product related logic and I will put all the product logic in there, so even my shop route here where I also work with products, I'll put that logic in here because all logic I have in my app thus far and this will change later but all logic thus far is related to products and therefore, I want to have it in a controller that only works with product logic. And if I later add some user logic, that will go into a user controller and maybe we even split the product controller into a user product and an admin product controller but for now, we'll leave it like this. So in that products.js file in the controllers folder, I now want to add that code here basically. So this middleware function we're executing here, this code for add product right where I just render this add product route, this can now go into there and the question just is how because this would be incorrect. Well in the end we want to link to this function and we want to link there from inside our route, so here we basically want to add a link to this controller function here. So therefore what do we need to do? We need to export this function in the product controller file and we can do this and we'll have multiple exports by using exports and then any name of our choice. Remember with this syntax, we can have multiple exports in one file easily. So here I will now export my get add product function here and the name is up to you but I name it like this because in the end this is what we do, we get the add product page. Of course you could also name it get add product page here but I will just describe what this in the end helps me do and it helps me get everything I need to add a product but you can name this to your likings. So now I get this product and I still have my function which receives the request object even though we're not using it here but we still get that, the response object which we are using and next because this still is a normal middleware function expressjs understands because I will now import get add product into my admin.js file and still using here on my route, this will not change, I just split my code differently. So with that, let's add an import, by the way we can get rid of this root dir import because I'm not using that utility anymore, you can also therefore delete the entire file or even folder but that's just a side note. So let's now import our controller and there, I will import products controller by requiring my controllers folder and there, the products file. So since I'm doing this from inside my admin.js file here, I need to go up one level until I'm in the root project folder and then I go into the controllers folder and then I pick that products file. So now products controller bundles all the exported functions and right now, this is only one of course but it will become more. So back in admin.js here on this route where I want to use that, I can now simply say products controller.get add product. And we don't execute this function so please don't add these parentheses, instead we just pass a reference to this function. So we're just saying or we're just telling express, the express router that it should take this function and store it and whenever a request reaches this route, it should go ahead and execute it. Now we can repeat this for adding a new product, I'll cut this code and go to my product controller and simply add a brand new export and I'll name this post add product because this is what I do here, I post a new product I add the product therefore through a post route and this will be the function I just copied. Now the problem here is that we refer to products which is of course something we don't have in that file yet, so I should also go into admin.js and take my products array which I have there, cut it out of there and then in products.js in the controllers file, I will add this array and I'll change this later too but for now, let's simply add it here, products should be our empty array. Now back in admin.js, we can now also use our new products controller function we just added, post add product in exactly the same way we use get add product and of course we can now tweak our exports here, we no longer need to export products in our admin.js file because we no longer have that array in here. So instead I'll just export my router again as I'm doing it in the shop.js file. So with module exports equal to router, that is what we have here too and this just means that we now have to adjust the app.js file where we are importing this, there I'm importing admin data, now a more fitting name is admin routes again because now we just export routes and nothing else and of course that also means that here where we use that, we just use admin routes because we change that export. Ok good, so we get this set up, now let's also do the same for shop.js. There I also get a function which is related to products, so let's cut it and in products.js here, I will add a new export, exports get products like this. If I do this, I have my function here which will in the end return or render that page with my products and one important note, obviously products is now an array which is available in that file, so products here doesn't have to be extracted from anywhere and again we will change this. So this is now my finished products controller in shop.js, we can now remove these two imports because we don't need them anymore instead I import my products controller by requiring it from the same path as in the admin.js file and here for this get route, I will simply say products controller get products. With all that, let's save this and see if it still works. If I reload this page, this all seems to work and if I add my first book here, this also works. So it's still working as before but now we're using a controller. Now let's see what else we can do here.

**Lecture 98**

**Finishing Controllers**

We added our products controller in the last lecture, now let's also make sure we handle the 404 route with a controller. In theory this is of course is not required, it's totally optional but it's a good practice. So definitely pause the video here and try this on your own before we then do that together, make sure to decide which controller you want to use, a new one or the existing products controller and how you would extract the 404, here is the 404 page logic into that controller and connect the existing route to it, good luck. Were you successful? Let's try that out and for that, let's first of all identify where we are rendering that 404 route and that is in the app.js file here at the bottom. Now we can absolutely leave that code here, it's very simple and there's nothing wrong with it but to be in line with our other code, I also want to put that into a controller. Now it's clearly not related to products because we can have many features on our page and every path can fail or the user can enter any random path, so instead I'll create a new controller here and you can name this however you want, you can name it 404.js, I'll name it error.js and in there, I will export a function with exports and then I'll name it get 404 page or you get 404 to be in line with get products which I have in the other controller and now I will cut that function here, this middleware function from app.js and in error.js, this is what I will store here in get 404. This is my function where I return the 404 page and with this I just have to go back to app.js and in there, I will simply import my controller, so const error controller is imported by going to the controllers folder and there to the error.js file and we can now take that error controller here and down there on app use, I will simply put error controller get 404, just like this as a reference to this function. And now this is also in line with the other routes and with the other products controller. So this is now the controller being added and with that we get views and controllers. Let's move on to the model next.

**Lecture 99**

**Adding a Product Model**

With controllers and views added, it's time to care about the model. The problem with our model is that we, well we have a very simple one, we manage our products array here and a product is simply created on the fly as an object that looks like this. Now in the end, that product represents our data, we have products in our app, later we'll also have our things like a user but for now it's just a product. Still we can define a model for that and for this, let's create a new folder in our root project and name it models. Now all these names are exchangeable of course but this makes the separation really clear, we get controllers, we get views, we get models and that makes up the mvc pattern. Now in there, I'll add a new file product.js . Now please note it's not products here because I want to represent a single entity because in the end, our core data is a product. Sure we also have lists of products with which we work but the core thing that makes up the app is how a product looks like, which fields it has, does it have an image, a title, that is our core data. A list of products is boring, it's just well more of that type, a single product is what defines our app in the end or part of what defines the app. Now how this this model look like? This is totally up to you in the end, you can define this in which ever way you want, you can for example simply export a constructor function here, so a function which I name product and you call that to create new objects based on that, using an ES5 constructor function but if you're using next gen javascript as I'm doing it here, you can instead create a class. You can create a class named product that looks like this and this is now also exported and in case you're not sure what a class is, check that javascript refresher at the beginning of the course, I do explain it there too. Now here in this class, I want to define the shape of a product and for this, I'll first of all create the constructor function, So here I want to receive a title for the product which I'll then create from inside my controller, so here I get my title and you can name this title of course and I will then create a property in this class, so basically like a variable in the class you could say. You do this with the this keyword and then this title is equal to the title I'm receiving as an argument here and these names don't have to match and to avoid confusion, you can also name this t here. So now I'm creating a property in this class, this allows me to then create an object based on this class where I can pass the title to the constructor which we call with new and then this will get stored in the created object. But obviously I don't just want to be able to create objects with a title, I can do this with the current curly brace syntax too, instead here I want to be able to create my or to store my product to an array of products and fetch it and for this I will reintroduce my products array here and we will change this later when we use a real database but for now let's go with this approach and I will add a save method to my class here by calling save or by typing save, adding parentheses and then curly braces. So it's like a function, just without the function keyword. So this is now a method available in this class and in the save method, I want to store my product in this array and I can do this by reaching out to products and then calling push here just as we did it before in the controller and I simply push this because this will refer to the object created based on the class and that is exactly the object I want to store in this array. Now obviously I also want to be able to retrieve all products from that array and I also want to do that through my product model, however whereas save makes sense to be called on a concrete instantiated object based on product, I also want to have a fetch all method which is like the utility function you could say. This is not called on a single instance of the product because it should fetch all products and I don't want to create a new object with the new keyword with some dummy title just to fetch all existing products and therefore I will add the static keyword which javascript offers which makes sure that I can call this method directly on the class itself and not on an instantiated object and then in here, I will return this, whoops, products like that. Now this is the model finished, now let's move to the products controller file. There I will first of all get rid of products here at the top and also of products push down there because now I want to use my model, I also don't need that anymore, so that I got no products array related logic left in this file instead I will now import my class by adding a new constant, product and you can name this however you want but the convention is to use a capital starting character for classes and in the end, we do just import this class so I will add the capital character in this controller file too and I do import a class by requiring this from the models folder, from that product.js file. With that added, in post add product I will now create a new object based on this class blueprint and that is what classes are in the end, they are blueprints. So I will create a new product, a local constant with new product and there I will pass request body title and that simply takes the title I have here as a name on my input which is submitted. With this, we create a new product based on our class, now there's one additional thing that needs to be done though, I want to save that and I can do that by calling product, save. This will use that save method we defined and it will therefore for now push that onto this array. Now with that in get products, I also want to fetch all products. So I will create a new local constant, products and now I will use that static method because I don't want to create a new product where I would have to set up some dummy title because I don't create a product here, instead I just want to use product and call fetch all and this should give me all the products and now I have my products here and if I save this, it should now work. If I go back and I reload this page, I get cannot read property length, that makes sense because fetch all returns this product which is incorrect, it should just return products because we are returning this array, not some local property of this class, there is no products property. So after fixing this and removing this down there, now if I reload this, this works and if I now try adding a first book, this again works now using a model. And whilst this might look more complicated right now and it certainly is because we're just using our dummy storage here, this is great once you really got more complex models with more fields, with more methods and where you don't store them in some random array but where you got the whole database connection logic and so on too, you then put that all into your model and you don't have to care about it in your controller. And it actually simulate this by moving away from our array storage here and move towards a file storage at least, before we then later in the course also use a real database.

**Lecture 100**

**Storing Data in Files via the Model**

So let's make sure we can save our product to a file and not to this array here anymore. For this when we call save here, I want to save it to a file and of course in that file, I want to have all products, the old ones and the new one. So therefore first of all, we need to be able to work with the file system, so I will import fs from the core fs module. Now that file should then also be created in special path, so I will use the path tool, the path module to construct a path that works on all operating systems. Now here in save, I will then create my path and I will do that with path join, so using the path core module, whoops, and I name this p so that I avoid namespace clashes and the path should be my root directory and we actually still have that helper function but if you deleted it, you can of course recreate it as we did before or simply copy this code here if you want, you also find it attached to this lecture again in case you deleted it. I will simply copy that logic here and move it in there but of course I could absolutely simply use my helper function I created there. But this is of course just the root directory, in there I will have a new data folder and I will create it ahead of time so that we don't get permission issues. So I'll have added the data folder to the root project folder and in that data folder, I want to store my file, so that will then actually be a file and that should have a name of products and I will give this an extension of json because I want to store my data in json format. Ok so that's products, now to store a new product in there, first of all I need to get the existing array of products, so I will first of all read that file. So let's use fs read file and this reads the entire file content of a file and by the way for very big files, there are more efficient ways because you don't want to read them all into memory before you work with them, you can read them as a stream then, there is such a function, you can create a read stream with this function but we can read the entire file here, this is ok and I will read the file at this path, p which is that file I'm interested in and then I will do something once I'm done reading it and there we either get an error or we get data, so we get the the file content you could say, there will be a buffer though. Now let's log the file content here and let's see what we get if we now call save. If I go to add product and I'll type something here, it crashes and it crashes, if you the scroll up because products is not defined in fetch all of course. In save however, it didn't throw an error but we get undefined here as you can see and we did get undefined because this file simply doesn't exist, there is no content in it therefore, you can see that here we've got no file with that name so reading it therefore kind of failed. And if I add an error here to print the error we get and I try this again, so let's go back to add product, click that and we scroll above this error message here, you see this is the error message we're now logging with this line and there we see no such file or directory and that makes a lot of sense because indeed, it does not exist as I just mentioned. So obviously if it doesn't exist, then I also want to continue and I can. I will simply check if we got an error and this will be null if we get none but if we have one then I simply want to create a new empty array because we have no existing products.json file, so we got no old products stored and otherwise I want to use the existing one. Put in other words I'll add a new variable here which I'll name products and initially this is an empty array and I will actually keep it as such if we do have an error but if we get no error and therefore I will add an exclamation mark here, so if we got no error, if this therefore is null so if we got no error, then I want to read the products from the file which I extracted. So therefore here I know that file content will be something, it should be the content of my file and since that is a json file, I will store it in json format there so then I will set products equal to json which is a helper object existing in vanilla nodejs, so you don't need to define this on your own and there we have the parse method which takes incoming json and gives us back a javascript array or object or whatever is in the file. So here I can parse the file content and that should work or at least we should try that. So now I know that product will be an array, either the one I read from the file or an empty one and therefore, we can now append our new product there, so I will call products push and push my new product which is this onto it. Now important, to ensure that this refers to the class, you should use an arrow function here because otherwise this will lose its context and will not refer to the class anymore. We have this setup though where I do use an arrow function, this should refer still to my class and therefore now I can push this onto this array, either to the new one or the one I read from the file and now the remaining work is that I need to save it back into the file. So again I will use the file system module and now I'll use write file and I will write it to the same path as where I read it from and I will put my json data into it, so again I will use that json helper object and now there is the stringify find method which takes a javascript object or array and converts it into json so that this has the right format, so there I will take this products array and convert this to json and then this gets written to the file. And here I also have a callback where I may get an error and I will simply log that error here to see if that works. With all that, let's go back, let's go back to the add product page and click this button. This error of course still exists but now if we scroll up, let's see if that worked. Here that is looking good and in data, we indeed see a products.json file which does contain that one product we created. Now that was the we have no old data case, let's now go back and add another product here and see if that also works. If I scroll up, well we also see it here I guess, this also worked out just fine. So this is working, we are able to read a file, append data to the existing data or create it if it didn't exist yet and therefore our data storage in the file seems to work, obviously it's a very basic storage but better than nothing. Now obviously we also want to be able to fetch the data from there, so in fetch all I also want to read this file here I will read the file at my path and I also will therefore get my error or the file content using arrow function here and this will then hold the data I want to use. Now if I got an error here, I want to return an empty array because then I got no products right, I always want to return an array because that is what fetch all expects but it should at least be empty in case of an error and of course you could also throw an error message and show one but here it'll just be the empty array otherwise and I don't need an else block because after return, we would finish the execution of this function anyways, so after this if block, I will return my parsed and that's important, otherwise it will just be a string, my parsed file content and this is important to keep in mind because this here is in the end retrieved as a text. So to return it as an array, you need to call json parse. So now I return my file content in a parsed form and therefore, I get rid of the return product statement here and I will now always return my objects or my list of products. So let's see this in action, let's save this and reload this page here, key is not defined, makes sense, I'm trying to read something from this path but that path is only defined in the save method. So for now let's copy it, we can of course refactor that but let's copy it for now and now if I reload this page, I still get this error and this can now be very hard to debug or to understand but do you know what's going wrong here? We'll fix it in the next lecture.

**Lecture 101**

**Fetching data from files**

I'm still getting the error that my products don't seem to have a length even though I'm fairly confident that this code should retrieve my products but what is wrong with that code? Well I am returning data here in both cases but keep in mind that this is asynchronous code. So my fetch all method here executes this line, executes this line and as you learned, it simply registers this callback in its event emitter registry to put it like this but then it just finishes with this function and this function itself does not return anything. These return statements here belong to this inner function here, not to this outer function, so fetch all does not return anything, it returns undefined therefore and hence in my view, in the shop.ejs file, if I try to access the length on my products, I try to access length on undefined and I get an error. So this is the issue and how can I fix this then? There are multiple ways of fixing this, for now I will simply accept an argument in fetch all and that's a callback and that actually allows me to pass a function into fetch all which fetch all will execute once it is done, so that the thing calling fetch all can pass a function it is then aware of being called which holds the data I want to return. Sounds complex? Let me show you how it works. So I will receive a function here, this argument will hold a function and therefore instead of returning an array, here I will have a callback where I pass an empty array, so I execute this argument as a function to which I pass an empty array and I will do the same down there, just not with an empty array but here I will have a callback where I pass my parsed json data. So I do call this callback and this allows me to go to my controller where I do call fetch all and there, I now simply have to pass in a function where I know that I eventually will get my products, like this and therefore I don't need to store it here because this fetch all function will not return anything. Instead here I simply create my own callback process and I render in that function I pass to fetch all once I know that fetching all products is done and I receive the products here because that is exactly the argument I passed to the callback in fetch all because the callback argument here will refer to this anonymous function I'm passing into fetch all. It's the same logic read file uses here, just that we didn't defined read file on our own but read file also takes a callback and here we pass in the function read file should execute for us once it's done. We do the same just that we write both sides now. We have fetch all and fetch all takes a function it should execute once it's done and once it's done, we get the products, thanks to our own implementation of fetch all and we then render our response with those products. And with that if we save that and now we reload this page, we do see the products here and we do of course see all the products we had in the past as well as any new products we add and there is a little styling page I see. But this is how this works and now it's still not a database but it's better than this in array storage and it already shows us why we might want to use such a model. It's not super complex code but it clearly is code that belongs to this data, to this model and therefore it's outsourced into its own separate file. I'm still getting the error that my products don't seem to have a length even though I'm fairly confident that this code should retrieve my products but what is wrong with that code? Well I am returning data here in both cases but keep in mind that this is asynchronous code. So my fetch all method here executes this line, executes this line and as you learned, it simply registers this callback in its event emitter registry to put it like this but then it just finishes with this function and this function itself does not return anything. These return statements here belong to this inner function here, not to this outer function, so fetch all does not return anything, it returns undefined therefore and hence in my view, in the shop.ejs file, if I try to access the length on my products, I try to access length on undefined and I get an error. So this is the issue and how can I fix this then? There are multiple ways of fixing this, for now I will simply accept an argument in fetch all and that's a callback and that actually allows me to pass a function into fetch all which fetch all will execute once it is done, so that the thing calling fetch all can pass a function it is then aware of being called which holds the data I want to return. Sounds complex? Let me show you how it works. So I will receive a function here, this argument will hold a function and therefore instead of returning an array, here I will have a callback where I pass an empty array, so I execute this argument as a function to which I pass an empty array and I will do the same down there, just not with an empty array but here I will have a callback where I pass my parsed json data. So I do call this callback and this allows me to go to my controller where I do call fetch all and there, I now simply have to pass in a function where I know that I eventually will get my products, like this and therefore I don't need to store it here because this fetch all function will not return anything. Instead here I simply create my own callback process and I render in that function I pass to fetch all once I know that fetching all products is done and I receive the products here because that is exactly the argument I passed to the callback in fetch all because the callback argument here will refer to this anonymous function I'm passing into fetch all. It's the same logic read file uses here, just that we didn't defined read file on our own but read file also takes a callback and here we pass in the function read file should execute for us once it's done. We do the same just that we write both sides now. We have fetch all and fetch all takes a function it should execute once it's done and once it's done, we get the products, thanks to our own implementation of fetch all and we then render our response with those products. And with that if we save that and now we reload this page, we do see the products here and we do of course see all the products we had in the past as well as any new products we add and there is a little styling page I see. But this is how this works and now it's still not a database but it's better than this in array storage and it already shows us why we might want to use such a model. It's not super complex code but it clearly is code that belongs to this data, to this model and therefore it's outsourced into its own separate file.

**Lecture 102**

**Refactoring the file storage code**

Now we added that file storage logic to our model, let's now fix this a little bit or improve the code in the model, we're reusing some code and if we reuse code that always screams for some refactoring and indeed, that is what I want to do. I will create a helper function and I'll store it in a constant and I'll name that helper function get products from file and this helper function will do this path construction here for me and it will also read the file, so it will basically do everything I do here in fetch all, like this and I will even get my callback here because I do execute that here and ultimately return this because the issue of this process taking some time and the need to inform the caller of this function about when it's done hasn't gone away. So I still use the same pattern of having this helper function which receives a callback which it executes once it's done reading the file. With this, get products from file is really just all I execute here in fetch all, so I simply just call this and forward the callback. Now for save however, it means that here I can also call get products from file but here I don't forward any callback because instead I have my own logic here, I do retrieve my products there and that's essentially the same logic I do have in get products from file, I either return an empty array if I have an error or I parse my content, here we did it the other way around but in the end it will result in the same result. So I'm only interested in this logic here then, you can remove that and instead just take this code here and then create a new anonymous function where I know that I will get my products because this again is the callback function, it is the function I will pass as an argument to get products from file, so it is what will get called here. So this function will get called in, will receive an empty array or the array with the data and I should by the way add a return statement here to make sure that we never execute this code after having executed this code, that was an error we had in the code before. Alternatively you simply wrap the other code in a else statement, either of the two, so here I'll use the if else approach. But now with that the callback will get executed and will get this array of products and now here in save, I do have this anonymous function which receives the products and in this anonymous function, I will then put in my code where I append a new product and make sure to always use arrow functions so that this never loses its context and always refers to the class and therefore to the object based on the class and then I write to the file, this means I can get rid of this line too. And now we get a slimmer version because we're reusing code but it still should work if I reload here, loading product seems to work and if I add a second book even though it's the fourth one, this crashes because p is not defined. Yeah the path here in save, that is a problem the path is not defined because I'm only defining it in my helper function now. Now there are various ways of fixing that, one of the easier ways is to simply create this as a global helper constant here, p so that I can use it in the entire file, you can of course also go for different solutions and pass it around in the functions but now the path should be available in the save method and in get products from file. And with that if I now reload and try this second, actually fourth book again, this now works. So now we are able to elegantly work with our products, store them in a file, fetch them from there and all of that through a model. And that is the MVC pattern implemented into our project.