**Lecture 58**

**What is Express.js?**

* It is a 3rd party library used to outsource functionalities like parsing requests etc.
* writing all that server side logic is pretty complex, just remember what we had to do to parse an incoming request. For extracting the body, we manually had to listen to the data event, to the end event and then create a buffer which we in the end converted to a string and this was just one type of data we could get. If we get other kinds of data, like for example we get a file or differently structured data, then we would have to write new logic. Now expressjs helps us with that, it actually doesn't have a built-in way of handling or parsing that data but it makes it easy to install another package that can easily be hooked into our project that will then do the parsing for us
* We in general don't want to care about all these nitty gritty details, we want to focus on our code that defines our application, so the thing that really sets our application apart from other applications, our unique selling point you could say and we do use a framework for this, for all the heavy lifting.
* A framework is basically a set of helper functions but also a suite of tools and rules with which we work, so basically we have a clearly defined way or at least some outline on how we should structure our application, our code and how we should work with that framework to write clean code
* Express.js alternatives 🡪 Adonis.js, Koa, Sails.js, Vanilla Node.js
* . But expressjs is by far the most popular and most often used one
* The great thing about express is that it's highly flexible and actually doesn't add too much functionalities out of the box but it gives you a certain way of building your application or of working with the incoming requests that make it highly extensible and therefore, there are dozens or hundreds and thousands of third party packages built for express specifically that you can then easily add to your node express application without having to configure a lot and this is probably the real strength of express and of course it also does add some nice features out of the box.

**Lecture 59**

**Installing Express.js**

* Install express as production dependency.

**Lecture 60**

**Adding Middleware**

* next() allows the request to go to the next middleware.
* Expressjs is all about middleware and you see a diagram here, in the end middleware means that an incoming request is automatically funneled through a bunch of functions by expressjs, so instead of just having one request handler, you will actually have a possibility of hooking in multiple functions which the request will go through until you send a response. This allows you to split your code into multiple blocks or pieces instead of having one huge function that does everything and this is the pluggable nature of expressjs, where you can easily add other third party packages which simply happen to give you such middleware functions that you can plug into expressjs and add certain functionalities but more on that later. So this is a core concept of expressjs
* The use method allows us to add a new middleware function, now the use method is pretty flexible, it accepts an array of so-called request handlers here and it has some other use cases too. Now one easy way of using it is that you simply pass a function to it and this function here, this function you pass to app use will be executed for every incoming request and this function will receive three arguments, the request and the response object as you already know it basically with some extra tricks learned though and a third argument which is the next argument.
* Refer code -- 01-adding-middleware

**Lecture 61**

**How Middleware works**

* . Send allows us to send well a response and actually this allows us to attach a body which is of type any, now let me show you what this could be. We could send good old html code here, just h1 tag, hello from express, like this. If we do that and we now reload this page here, we see hello from express, by the way one thing you'll notice is that if you open your network tab here and you inspect that request you got, you will see that under headers, the content type is automatically set to text html here. So this is done for you, this is another feature provided by express here
* . The send method by default here since we send some text here simply sets an html content type, you can still set one manually with set header of course, so you can always override this expressjs default but you can also rely on the default where the default response header is text html

**Lecture 62**

**Express.js – Looking behind the scenes.**

* Refer Notes
* Refer code -- 02-looking-behind-the-scenes

**Lecture 63**

**Handling different routes**

* Refer code -- 03-handling-different-routes.
* , Now what happens if I for example enter /add-product? We still see hello from express and we still see I'm in another middleware, so this middleware gets executed for both slash and add product because this does not mean that the full path, **so the part after the domain has to be a slash but that it has to start with that**. Now of course every route starts with just a slash and then we have different other criteria. So what we can do is we can simply duplicate this and add it before this middleware and add /add-product. Now why before this middleware and not after it? Because remember, the request goes through the file from top to bottom and if we don't call next, it's not going to the next middleware. Well I am not calling next here, so in the end if we have /add-product, this middleware will be reached first because top to bottom, add product will match this middleware and since I don't call next, this middleware will never get a chance of handling that request even though the filter here would have well, matched that request too. So here if I just add the add product page like this and I save this, you will see that on /add-product, we see the add product page and on any other path including random stuff or just slash nothing, you see hello from express

**Lecture 64**

**Parsing Incoming requests**

* Res.redirect(‘/’) 🡪 redirects to the path /.
* By defeault req.body attribute will be undefined. Because incoming data is not parsed automatically.
* For that we need to use body-parser package and register a middleware bodyParser.urlencoded() , for form inputs. For inputs like json , files etc, we have to use other middlwares.
* Refer code 04-limiting-middleware-execution-to-post-requests

**Lecture 65**

**Limiting middleware execution to post requests**

* Refer code 04-limiting-middleware-execution-to-post-requests

**Lecture 66**

**Using Express Router**

* Refer code 05-using-express-router
* The get(), post() methods etc use the exact match unlike the use() method.

**Lecture 67**

**Adding a 404 Error page**

* Adding a middleware at the last , it will be executed when other matches are not found.

**Lecture 68**

**Filtering paths**

* Refer code 06-filtering-paths

**Lecture 69**

**Creating HTML Pages**

* Refer code 07-creating-the-html-files

**Lecture 70**

**Serving HTML Pages**

* Refer code 08-serving-the-html-files
* \_\_dirname is a global variable made available by nodejs , which holds the absolute path of the folder in which this file resides.

**Lecture 71**

**Returning a 404 page**

* Refer code 09-returning-a-404-html-page

**Lecture 72**

**Using a helper function for navigation**

* Refer code 10-using-a-helper-function.
* path.dirname(arg) returns directory name of a given file
* process.mainModule refers to the main module which started the application , i.e the module created in app.js
* process.mainModule.filename gives the name of the file in which the main module was spun up, i.e app.js
* i.e in the end we get the absolute path of the directory which contains app.js

**Lecture 73**

**Styling our pages**

* Refer code 11-styling-our-pages

**Lecture 74**

**Serving files statically**

* Refer code 12-serving-files-statically.
* Refer notes.

**Lecture 75**

**Wrapup**

* You can filter request by path and method easily with app use by adding a path or app get, app post and that if you filter by method, like if you had app get, that the paths would then be matched exactly otherwise with app use, the path you passed would only be matched with the beginning of the url, the part after localhost. You also can use the express router package instead of app use, app get because this allows you to elegantly split your routes across multiple files since the router you export there can be added as a middleware function into app use in your root file.