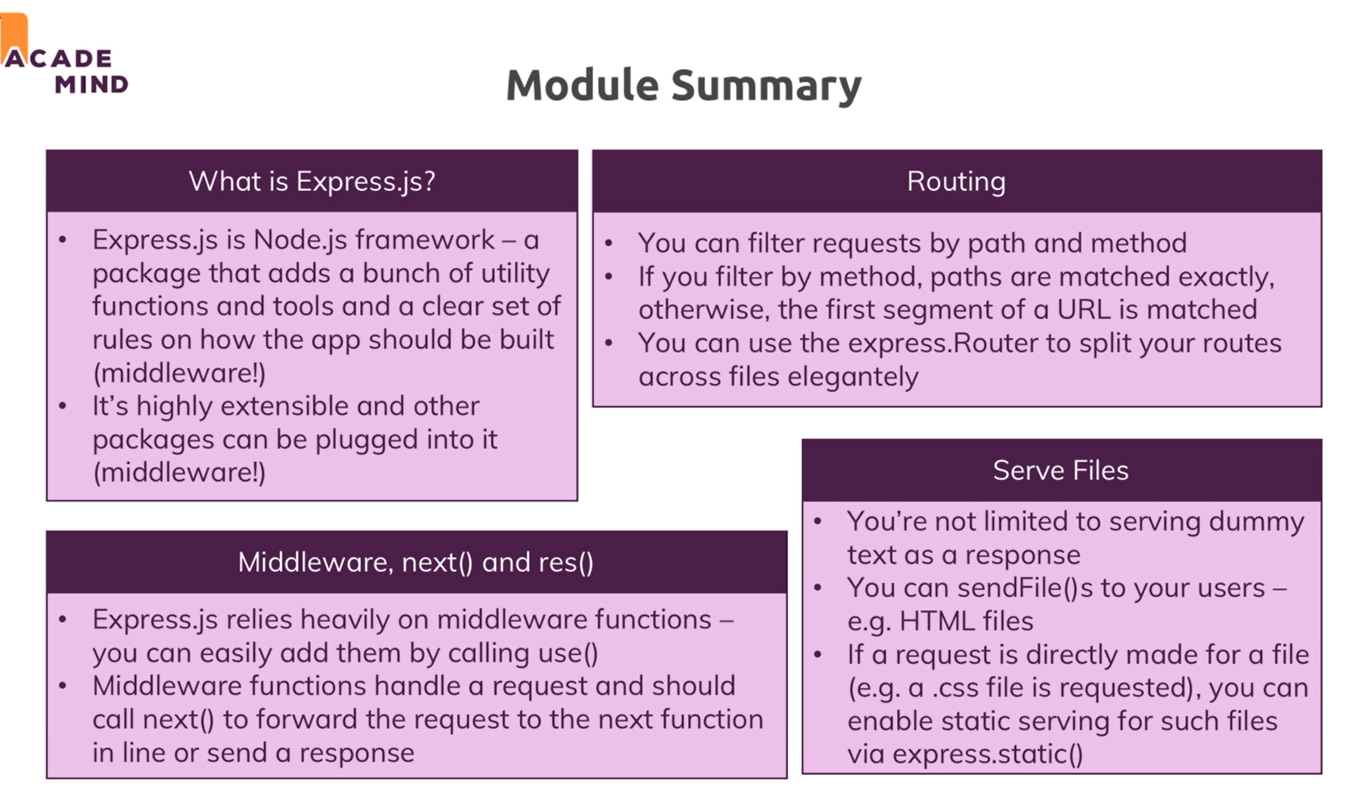
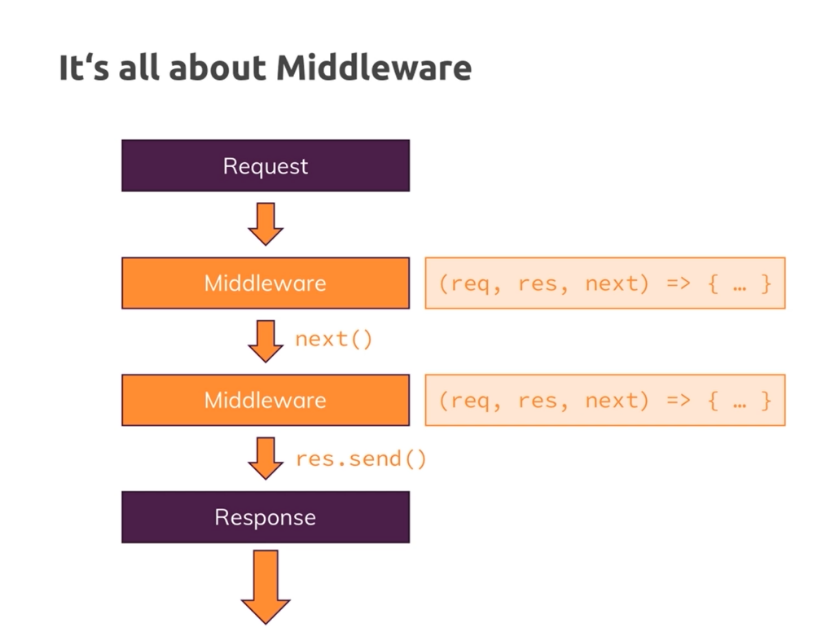
Express.js

**Table of content:**

* What is Express.js?
* Using Middleware
* Working with Request & Response
* Routing
* Returning HTML Pages

**Summary:**

1. **What is Express.js & Why we use**a. Writing all that server logic is complex, to extract body from incoming request we have to listen data event, end event and then create a buffer which we convert to a string if incoming data is string else for other types of data, we have to do other operations. With Express.js we can parse body in a simple way.  
   b. We use express.js (or other framework) in our application because we want to focus on business logic of our application not on nitty-gritty details  
   c. Framework: Helper functions, tools & rules that help us build our application.   
   d. alternative of express.js: vanilla Node.js, Adonis.js, Koa, Sails.js, ………
2. **Installing Express.js**a. install using: npm install --save express command.  
   b. to use it: import express,   
    const express = require(“express”);  
   c. to create express application,  
    const app = express();  
   => express module, export express as a function, this function has lot of thing to us. (lot of logic in app const contain)  
   d. this is valid requestHandler hence, we pass it to createServer  
    const server = http.createServer(app);
3. **Working of Middleware**express.js is all about middleware.  
   a. middleware means incoming request is automatically pass to the functions by express.js.  
   b. Instead of one requestHandler, we can have possibility of hooking multiple requesthandler until we send response.  
     
   c. we can use that using, use() its accepts an array of requestHandler.  
   d. if we write app.use((req, res, next)=>{}); between creating a express application(const app = express();) and creating a server (const server = http.createServer(app);), It will execute for every incoming request and it receive 3 parameters: req, res, next(it is a function which is passed by expressjs has to be executed to allow the request to travel next middleware)  
   e. to pass to next middleware we have to call next() else it will pause at that function.(request can’t continue its journey)  
   f. if we don’t call next(), we should return response.  
   g. to send response use, send(), res.send(‘<h1> hello world</h1>’); express automatically set header, if can also modify header.
4. **App.listen()**we replace:  
   const server = http.createServer(app);   
   server.listen(3000);  
   with  
   app.listen(3000); which internally call http.createServer now we can remove dependency of http module from our code.
5. **Handling Different Route**in app.use(), 1st parameter is path, and 2nd parameter is a callback  
   eg:   
   app.use('/add-product',(req, res, next)=>{

console.log("In the middleware of add product ");

res.send("<h1>Hello from Add Product page</h1>");

});

1. **Parsing incoming data** a. to parse body of incoming request we have to add a 3rd party package called as body-parser.  
   b. import body-parser and create a middleware for a body parser at the top. (reason top to bottom effect)  
   const bodyParser = require(‘body-parser’);  
   app.use(bodyParser.urlencoded({extended: false}));  
   c. now we can get request body using req.body in json formatted.  
   d. to execute middleware for post method only use: app.post() in place of app.use() (similary for get, add, update, delete)  
   e. order of execution is top to bottom. In case of get, post, update, delete passed url must be same as given url (exact match here)then it we execute but in case of use() exact match is not  
   f. if an user add any url, if our application don’t have and we have not used use() then application will give display error(in-valid route)
2. **Using Express Route (express.Router())**a. in a project we can not write entire code in a single. We have to segregate our code into different files. To-do so we had already learned **module.exports** but express has in-bulid route method **express.Router()** which is used for **out-sourcing routing** in other files.  
   b. **express.Router()** is a mini express app which is plugable into another express app.(which we can export it)  
   c. to use this concept, we create a folder (routes) in our node application and create files based on task assign to them. We export router from these files and import that into app.js file.
3. **Adding a 404 Error page**a. reason todo this is if an user entered wrong url or any url which is not defined in our application(and we have not used use()) then node application will give error(in-valid url).   
   b. to prevent this, we will write use() method at last of app,js in this we will return error 404 page with status 404.  
   eg. res.status(404).send(‘<h1>Page not found</h1>’)
4. **Filtering Path**a. we this when our url in add router start same part  
   eg. /admin/add-product, /admin/product  
   here, /admin is same. We can add /admin in use() of app.js  
   eg. app.use(‘/admin’, adminRoute)  
   and in app.js we will only use /product, /add-product.
5. **Html Pages**a. to send html file as a response, we use res.senfFile(); and we pass path to that html file using path module(core module of node)