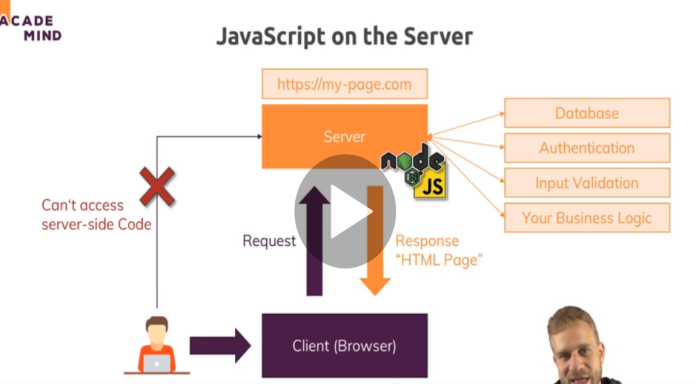
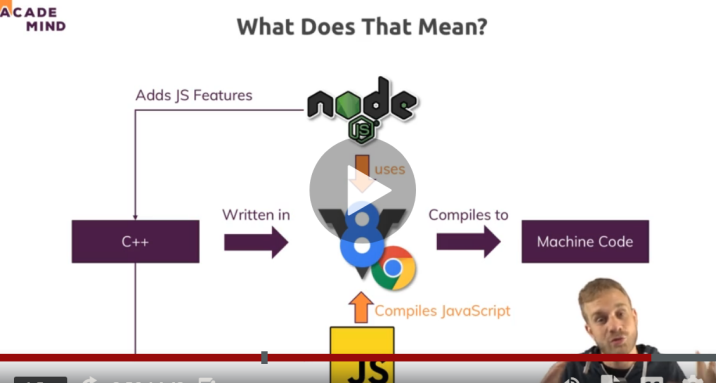
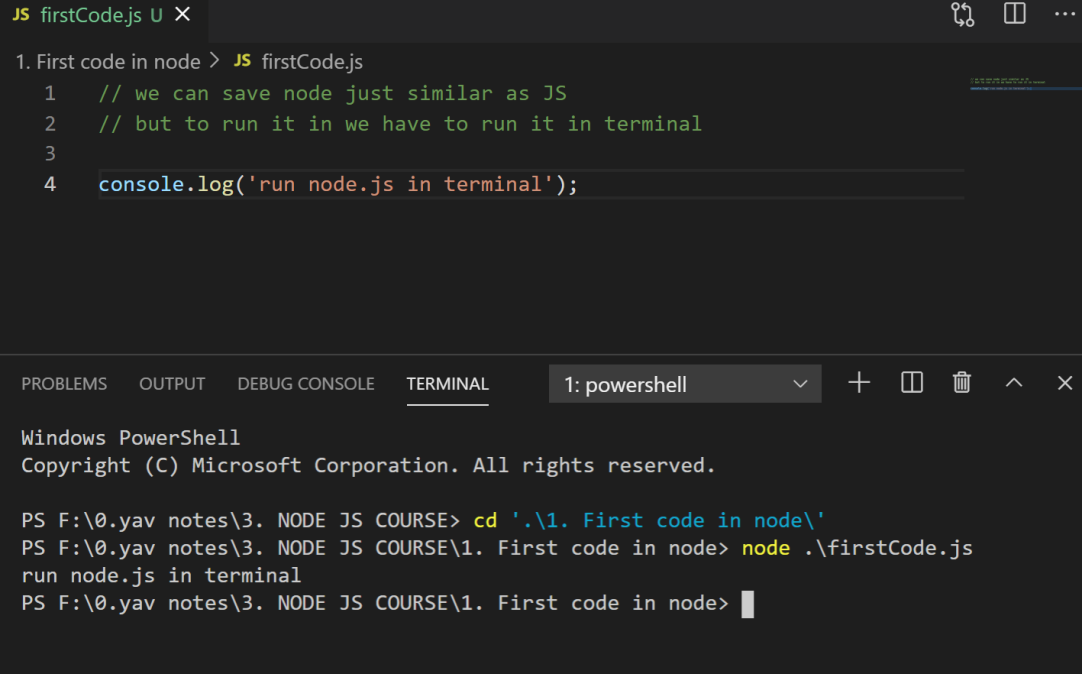
* **INSTALING NODE JS**

Node JS is the JS RUN TIME which add some features to JS so it can run of servers, so node js is just like other programming languages which can run on Systems (while js is for the browser purpose)

Node js use V8 which is Google compiler which takes the js code and convert into machine code in browser.

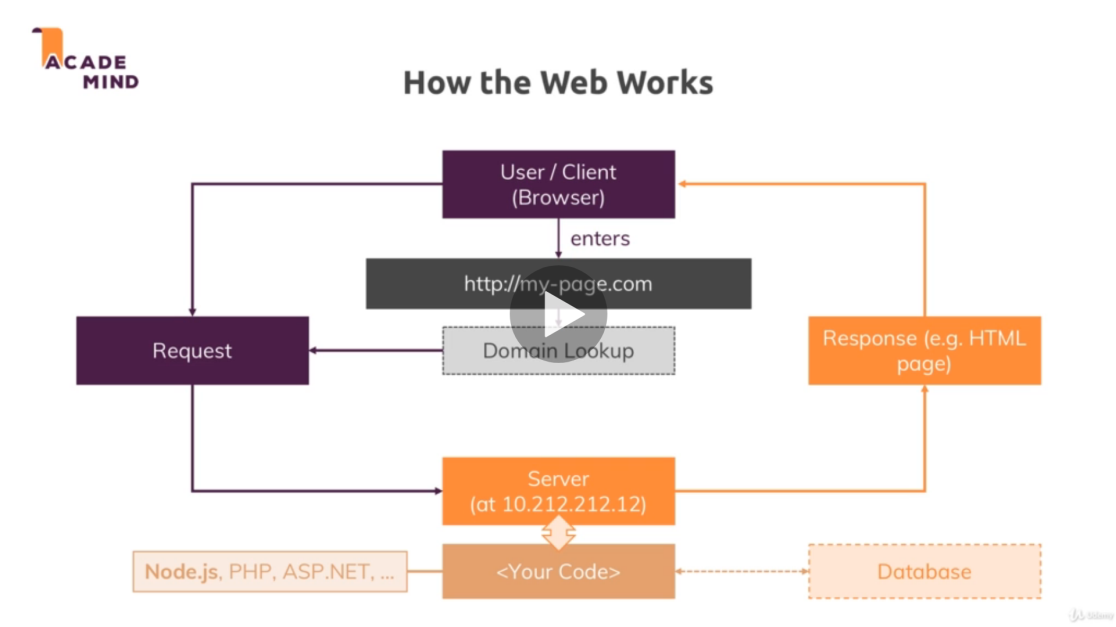


* Run node in vs code with terminal

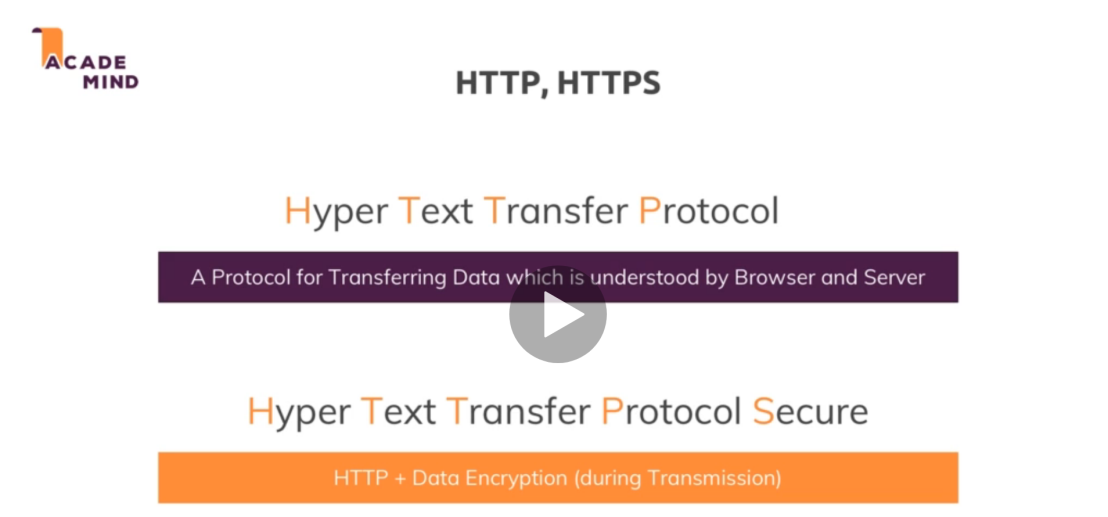


Q SO how node js is working in server side

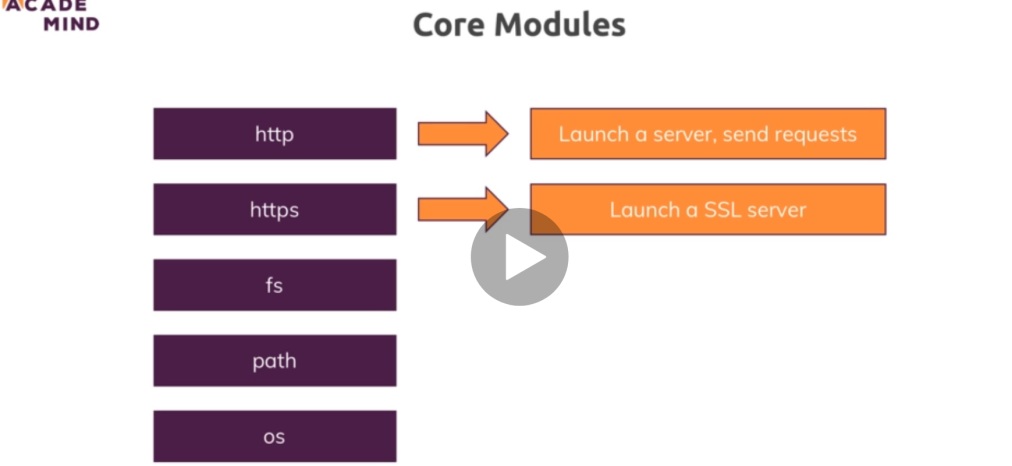
So when user enters url this url converted by domain lookup and sends the request to the server side where node js plays a work to handles a data from data base and give response back to the client in form of html page.



So to send from putting a request to getting it all the work done under standardized protocol which is HTTP and HTTPS but in https data is encrypted and secured.



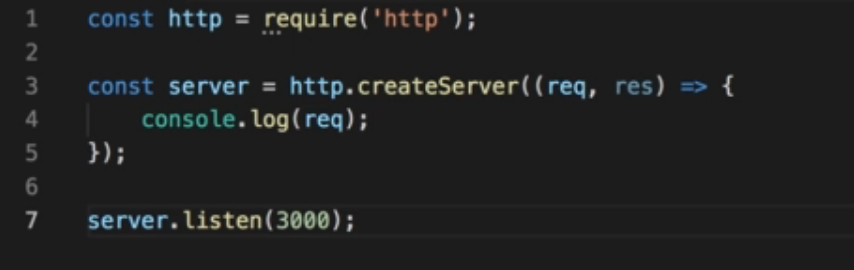
Some of the core module of node js which are most in use are



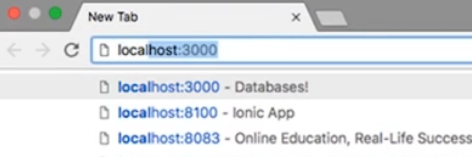
* How to write your own server in node Js

For that you have to get and save the path of the request some where which should be const for constant not changing 

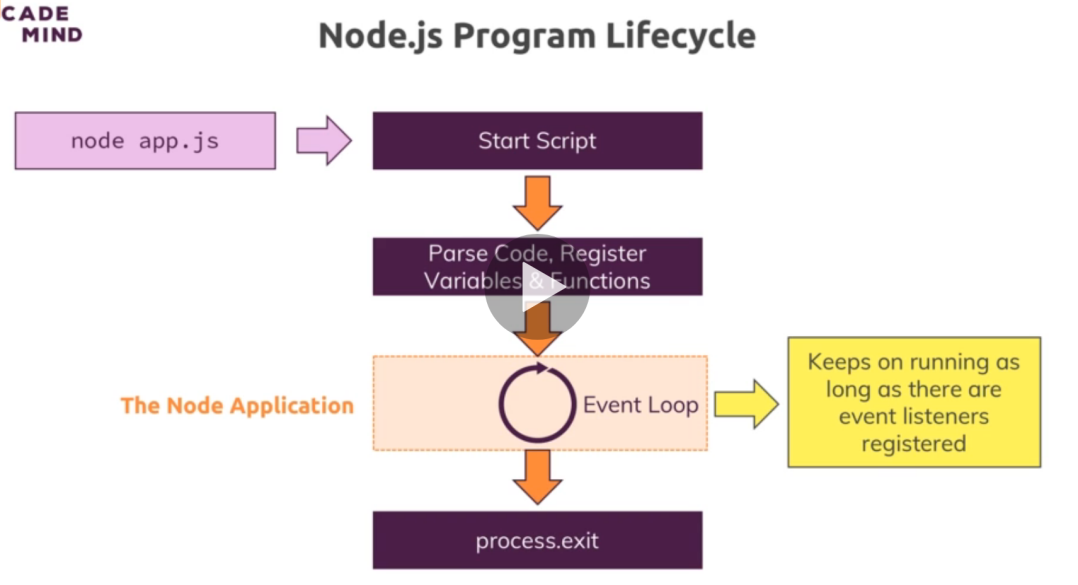
Since the path is not defined therefore we using http only( but the actual path starts with (./http))



Here above we get request from the required(http) which is the local host or our net server and call a method from that http to create Server and print the request of it and after saving in server we listen the server by passing the port value greater then 1000 which is safe and her we do not pass other argument hostname which by default pass the local host and this keeps on listening request ( so when write localhost 3000 ) in our browser the it will listen to the browser. And you can check it by node in vs code

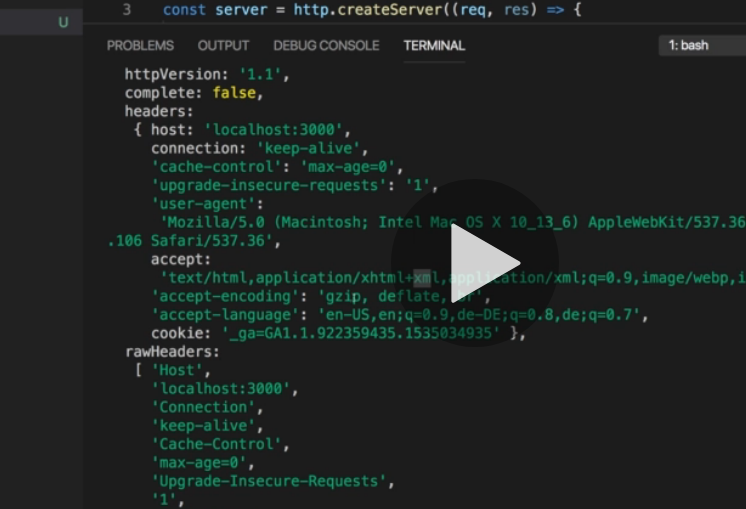


Node life cycle is when we call the node (file) it start scripting then parse and memorize the code and go to event loop and keeps on running until it listen the server it keeps on running so client can see the data so to exit it we use process.exit (when client do it.)

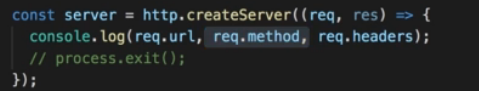


* Request in node js

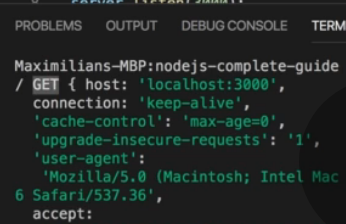
So when we console the request we get whole bunch of data, variables and methods etc



But we want only few to understand those are , request url, method and headers (headers contains the meta data)



So here is the output in terminal

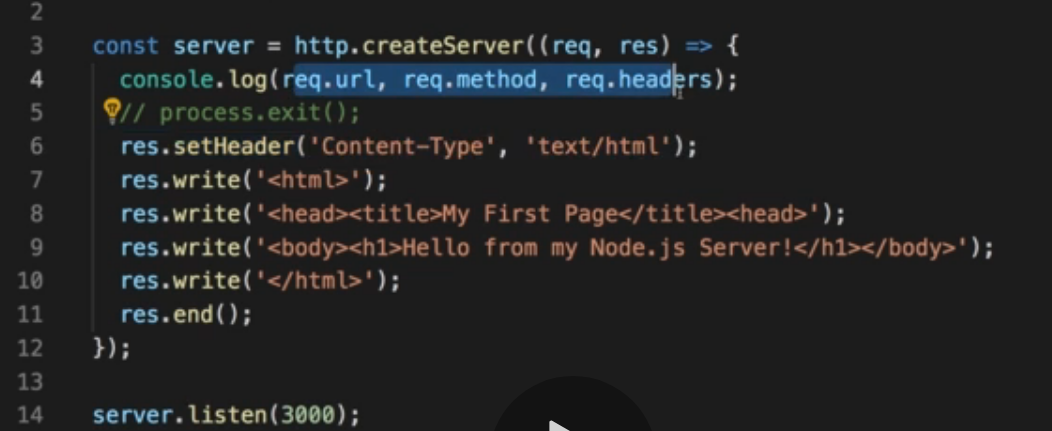


Which is (/) for the url because it is localhost (get) is for method and ( host : 🡪 below) is for header

* Response in Node js

So we can give response to the user/client side in node js can set header by res.setHeader(‘Content-Type, ‘text/html’) // so inside it we have to pass content Type.

We can also write a html code by our own inside the Js by res.write(); and when we have to push the our html code to the client that we have written have to use res.end() to end the response after this we can not write any res.write() html



On both requests and responses, Http headers are added to transport metadata from A to B.

The following article provides a great overview of available headers and their role: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>

// her we create a local server

const http = require('http'); // get the url request

const fs = require('fs'); // fs enable us to work with the file system

const server = http.createServer((request,response) => { // crete server from teh url req. and its response

    const url = request.url;

    const method = request.method;

    if(url === '/') {

        response.write('<html>');

        response.write('<head><title> response in node </title></head>')

        response.write(`<body>

        <form action="/message" method="POST">

/\* the action in the form will move the page to that/message\*/

        <input type="text" name="message">

/\* name message will be shown on the network  menu doc \*/

        <button type="submit">Send</button>

        </form>

        </body>`)

        response.write('</html>')

        return response.end(); // we have to return the after response end so it will not go down further because we do not write after response end

    }

    if(url === '/message' && method === 'POST') { // it will run only the above condition run

        fs.writeFileSync('message.txt','Dummy'); // it will create the file with of .txt

        response.statusCode = 302; // it sends for redirection 302 will redirect

        response.setHeader('Location','/') // it will locate the heder back to (/) home page

        return response.end();

    }

    console.log(request); // it keeps on running in the event loop

    // process.exit() // it is used to quit the server

    response.setHeader('Content-Type','text/html');

    response.write('<html>');  // it will write the in the response in form of html

    response.write('<head><title> response in node </title></head>')

    response.write('<body> this line written by response.write inside the node</body>')

    response.write('</html>')

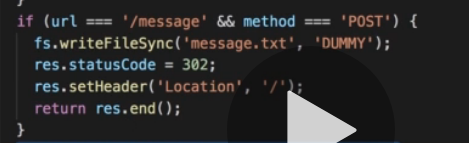
    response.end();

})

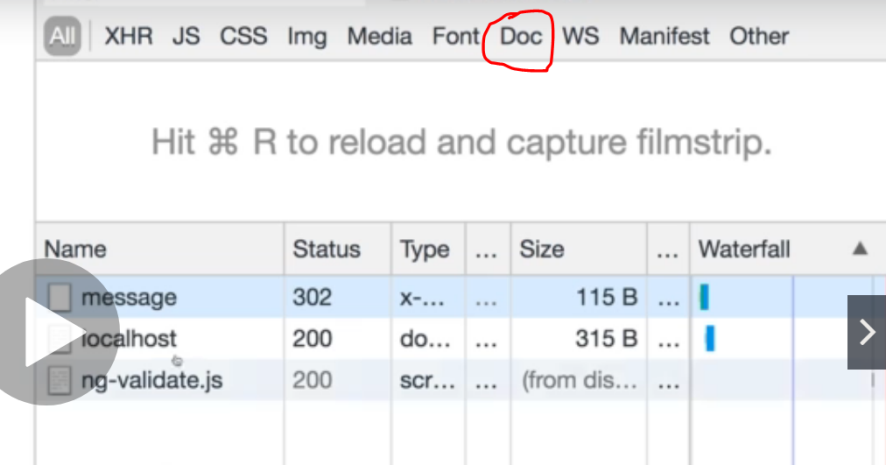
server.listen(3000); // it will listen the server continuously with the port of 3000

// and the host name as local host by default if we do not pass so

// so when we write localhost 3000 in our browser it will console in our node



The above code set the status as 302and move page to back /



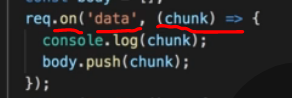
* Streams and buffer in node Js and the data

We are using streams and buffer so we can get the data that we stored in message above

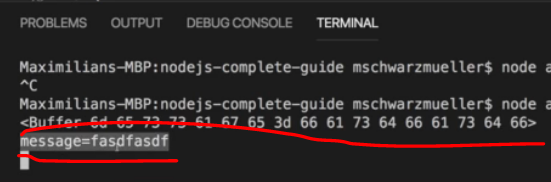
Streams are the objects which lead you to read data from a source or write data to a destination in continues manner. We can say stream as live stream or streaming the movie and not downloading it. Stream will provide us data continuesly while we watching it

Buffer is the temporary chunk of data that is being transfer from one place to another in small amount without waiting the whole data to load.



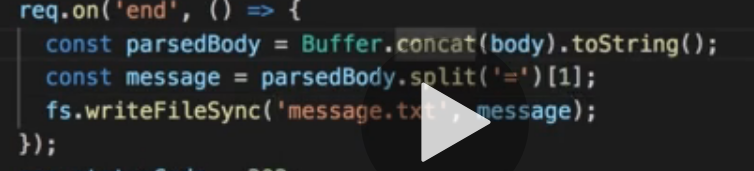
So to get the data have to listen the data first by event listener which is written as .a()  here the event is data and the we run the method on it when ever it pass the data and pushing in body array so we can use with buffer.

So we have to write another event that is (end) so when all the data passed at the end we can use the buffer as bus stop and convert it to string so that we can get the data as string



So the first red line is the console of first while (2 is the console of 2nd line)

The data is saved in key value pair which is message as key and = fasd----f as value

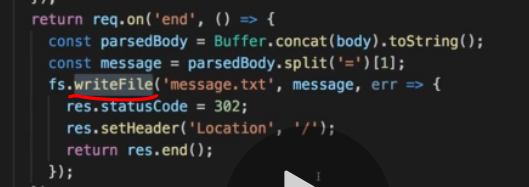


So above we storing our data at 1 index of split into message and writing a file containing message in our system.

* Event driven code and blocking and non blocking code

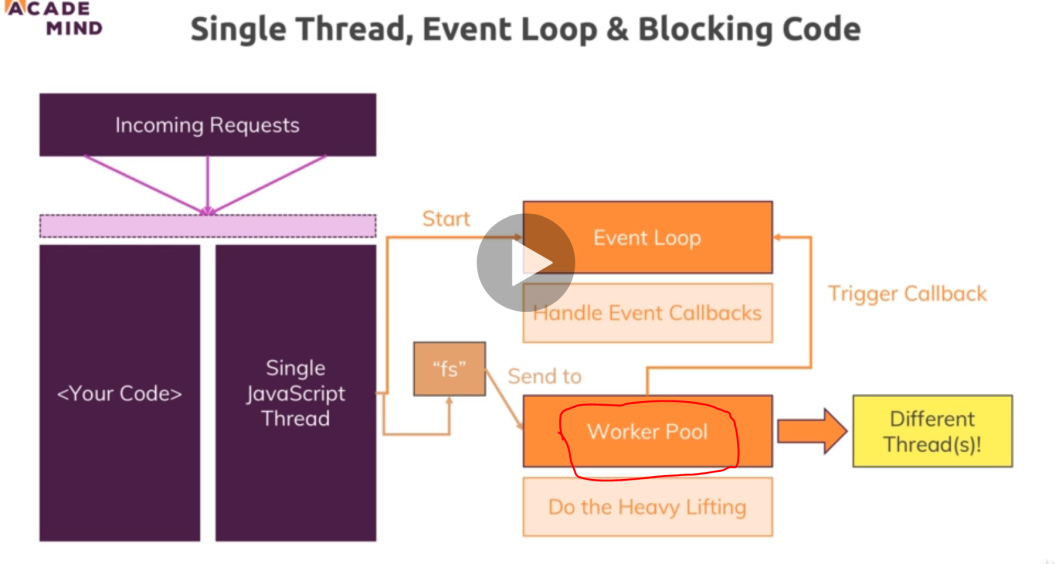
So event listener store that into registry because it is async in nature and run after all the code executes

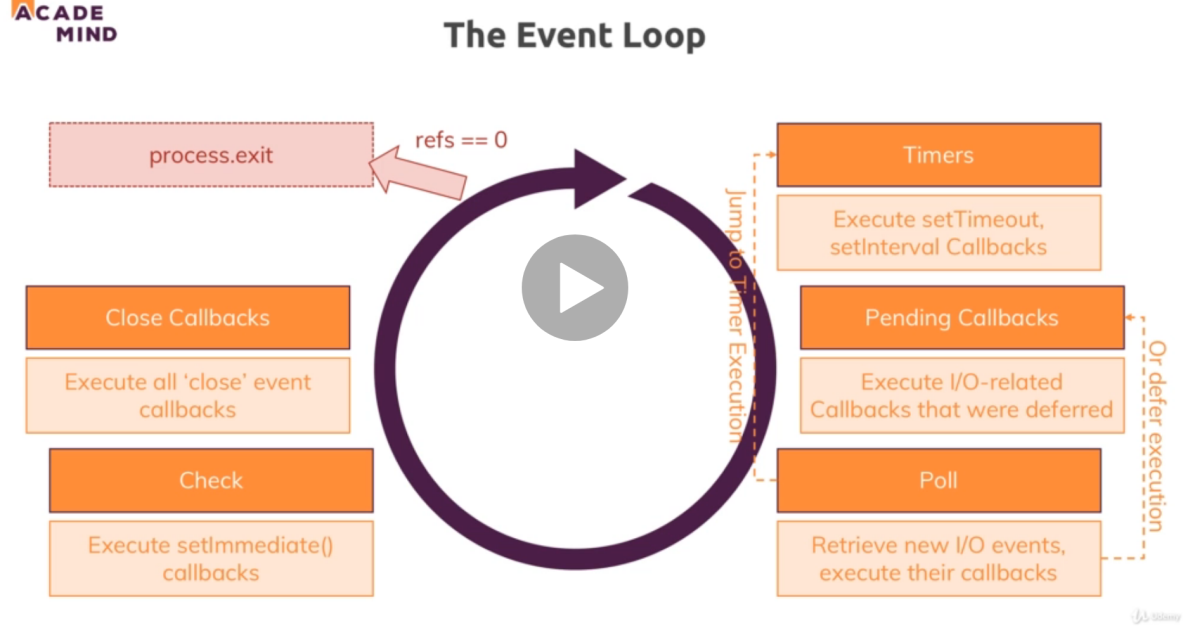
The differernce between .writeFileSync and .writeFile is that .writeFileSync is Syncronus and if it is large file to write then it will block the code below so it is good to use .writeFile because it is async and run after the code completion.



Node js just like java Script single threaded execute one line at a time and the event loop run the sync function first and then Async function after completing all line of code so the set time out will run from the call back queue when the call stack is empty

But in node all the methods and object made by (“fs”) will run inside the worker pool so that will do the heavy work of file



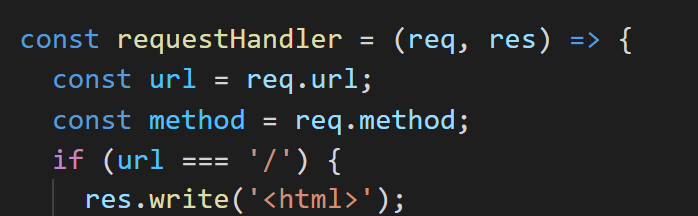


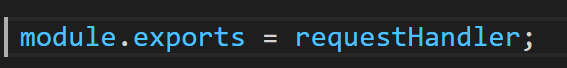
So event loop work on all this and run continues until (process exits)

Poll takes the i/o callbacks and send postponed callbacks to pending callbacks the timers will run simultaneously and jumping from poll to timers and timers to poll according to which done first

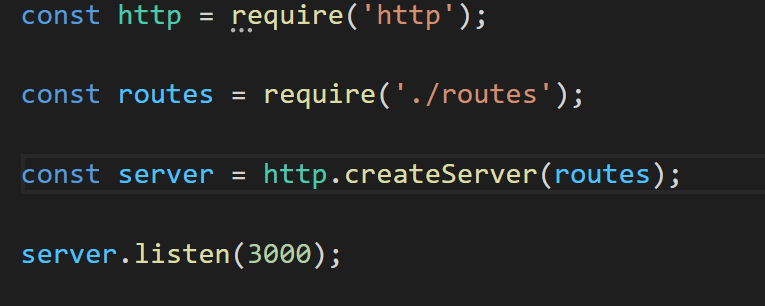
We can connect to Node js files with module.exports in

Second file

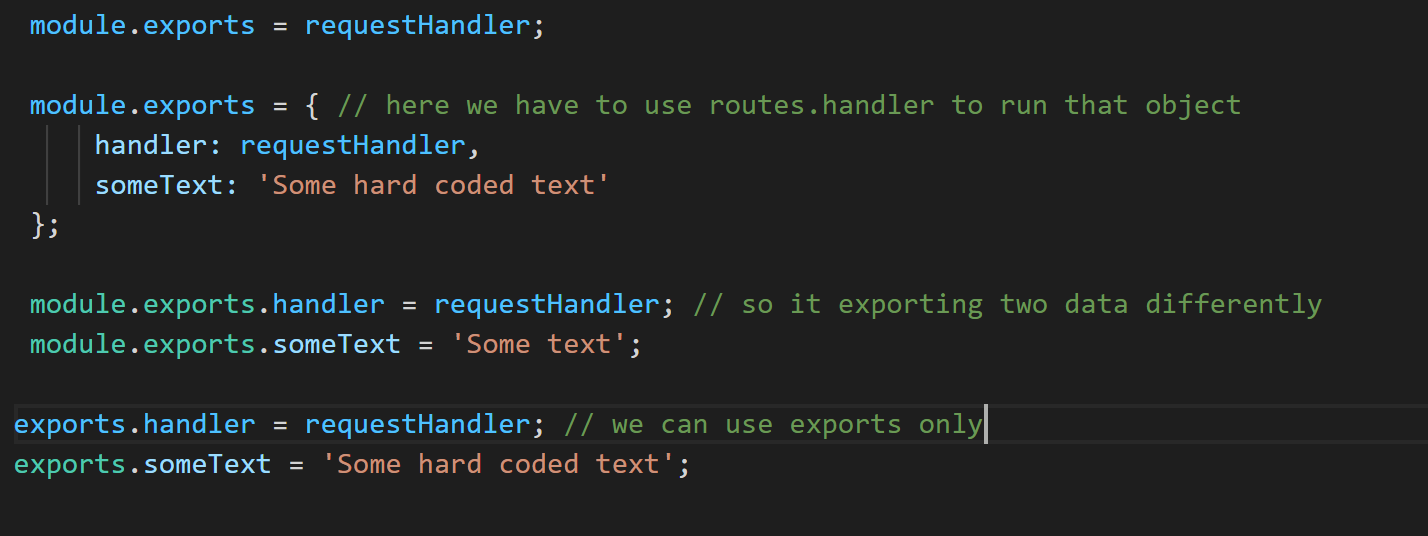




First file is

 so when we pass routes then it passes the request and response inside as module .export

We can also write module export as



Useful Resources & Links

Attached, you find the source code for this section.

Useful resources:

* Official Node.js Docs: <https://nodejs.org/en/docs/guides/>
* Full Node.js Reference (for all core modules): <https://nodejs.org/dist/latest/docs/api/>
* More about the Node.js Event Loop: <https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/>
* Blocking and Non-Blocking Code: <https://nodejs.org/en/docs/guides/dont-block-the-event-loop/>

**Use of npm in Node JS**  
npm is used to install 3rd party packages in our project and those packages are available in npm repository

And we can install a package using **npm install** which will get install from npm repository and if we write –g it will installed globally

So when you write npm install it will install the node module folder

IMP/ => **How to setup express with node**

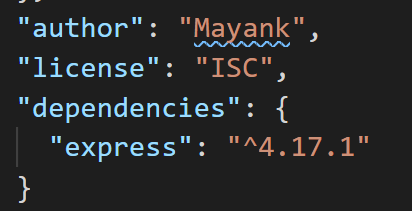
So first we have to write npm init in terminal using git terminal or vs code terminal terminal should be opened in the project

After creating it a json file is created with the name (package.JSON) maybe so to use the npm scripting commands you have to write that command and what it will do inside the script of package.json like npm start you have to give the name of that file to start



After that to install express you have to write npm install --save express

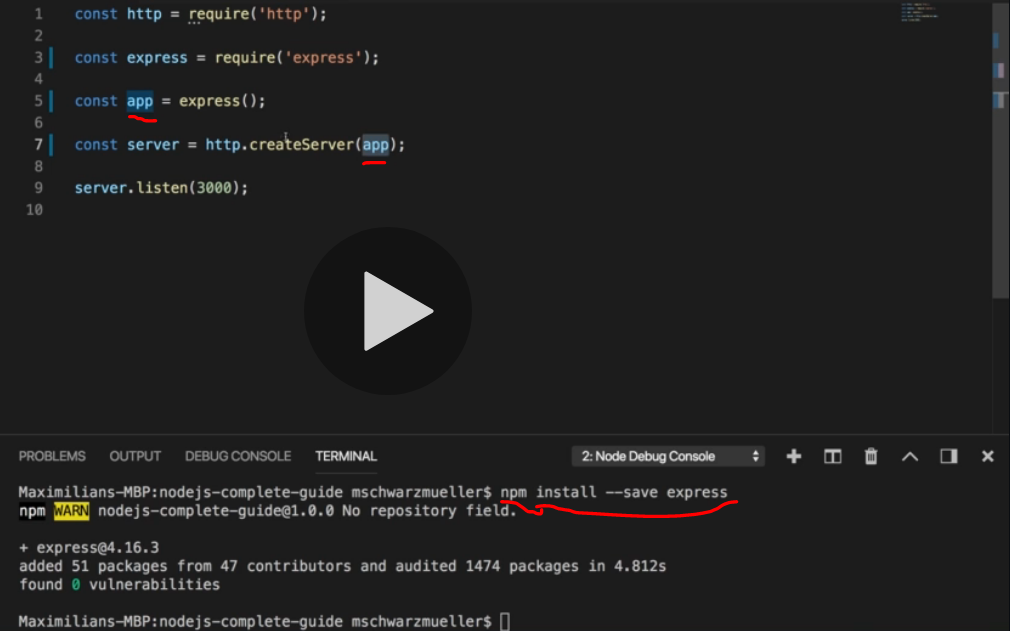
 we are doing it because we are not installing it in separate file instead we are adding dependencies in our package.json file



Useful link of installing express = <https://expressjs.com/en/starter/installing.html>

* EXPRESS JS

So we have to use express js with node js to buid the server side and server side api’s

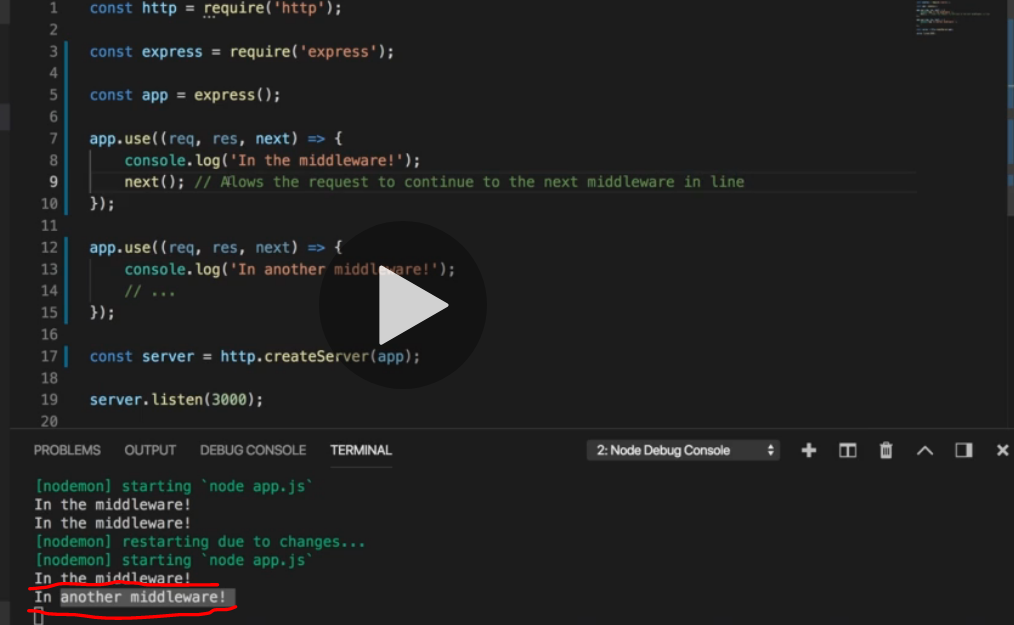


Here we are saving our express in our old file so we can use express from it by require so after saving in const we are saving it to an app as a function so that we can call it in server

So npm install –save will add the dependencies in our package.json file.

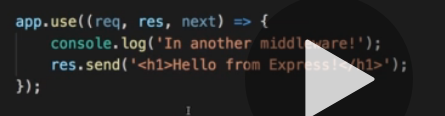
So the use of middleware In express

We can add multiple function and methods in middleware so that we do not have to write whole code at one function instead we can write multiple function in it



So that we can use app as function which containing request, response as we see earlier but the net is used for calling the next function of app if we do not write next it will not run the another function

We can also send the data to server which is by default in html

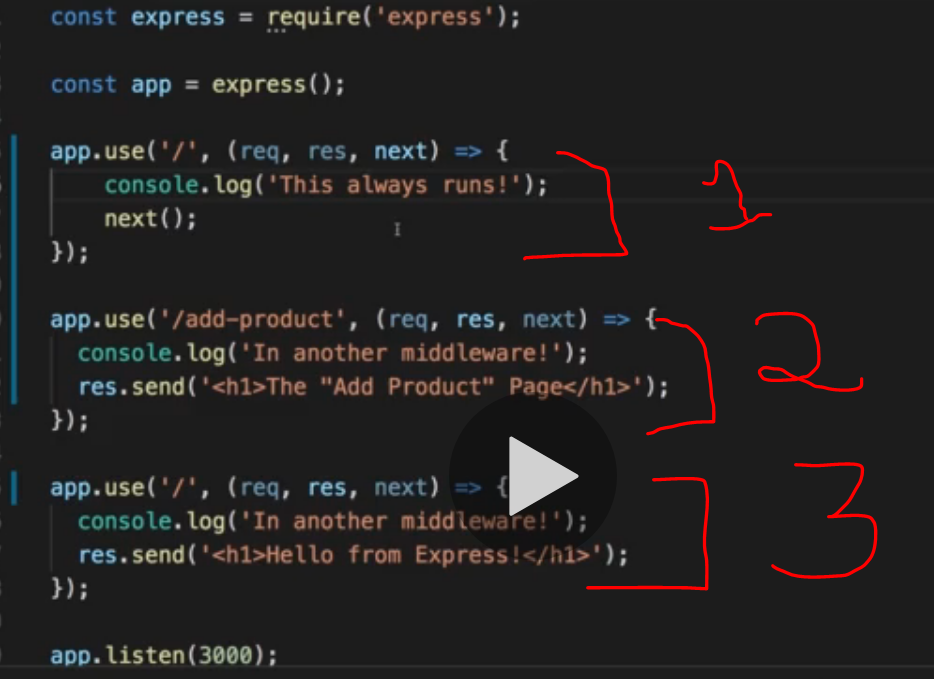


Res. Send is saving a lots of code because previously we are using a buffer and hole lots of code which we do not have to write because it is already written in the express js you can see it on their git repository of express code.

So express js will sort the repetitive coding part which is used more often so we can do our real operations

So we can use app.use in different ways if we do not write anything in that means it is applicable for all and we have to do next() to perform different operation which is below it but if we use app.use() with some path that will run in any case for that path only so we do not have to use next but if we write (./) only it means it applicable to all and it is default too.

 we can have more then 1 callBacks



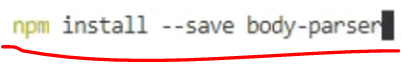
So the first one is applicable to all and we have to write next() to run the next on the list

The second one will run for the /add-productpage so we are not writing next because we do not want it to go down we only show that response only

And the third one is for all, so if we write add-product/something it will still open the product page

* Form Handling in express js

Form Handling is done by body parser in js from which we can parse JSON type of data, url encoded type of data and many more ( url encoded data is smoothing that is encode in form like your name)

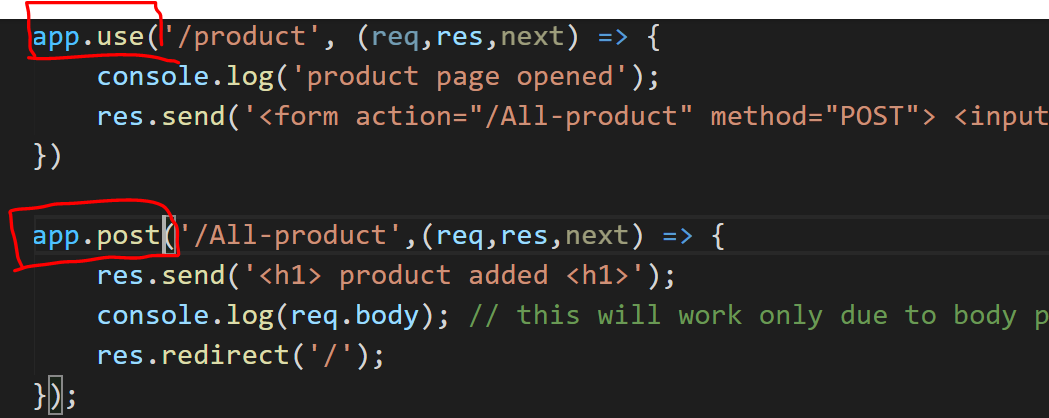
So to enable body parser you have to first install body parser by it will save the parse file in our package dependencies



Just like we are using body parser by require and using it in the app where we are using url encoded method with seting default values as false.

Which can run  so from then you can see your object

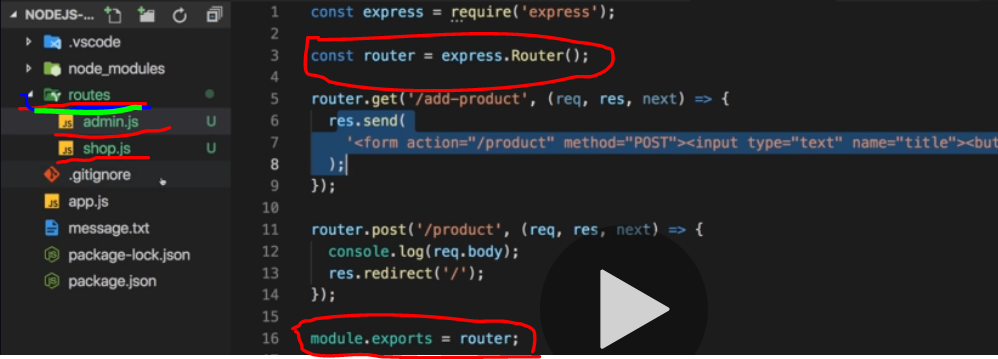
We can also use get,post,delete etc in place of app.use



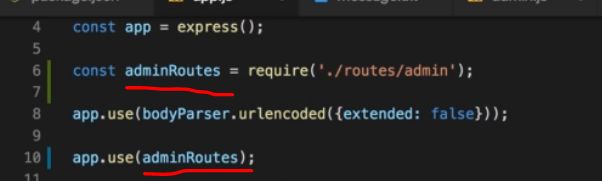
So the advantages of writing get,post ect instead of app.use is that they will run no matter where is the position of them and they will run

* OUT Source the routs

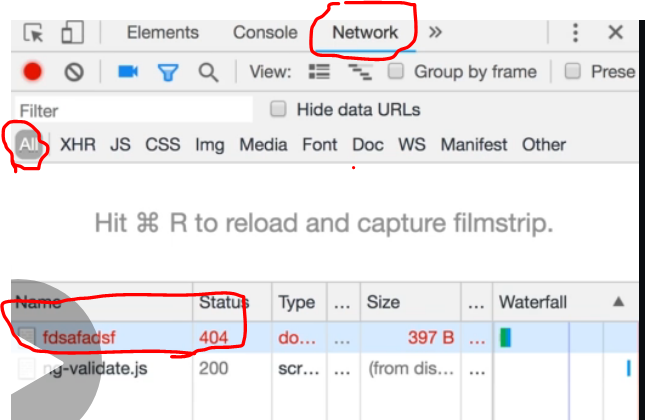
We can also out source the routs which means nothing but creating all the routs which we are creating as app.use,get or post in the different file that we can name it as routs and exporting them so that we can use it in our app.js files



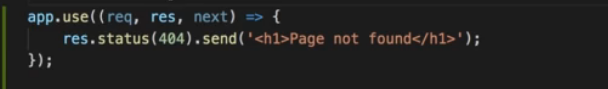
Here we are exporting routs in our admin,shop.js so we can export it into our app.js file



So their we are exporting admin.js by require in adminRoutes so that we can call it inside our app.use() which work same as it working so that we can listen our server by it.

 http://localhost:3500/fdgdfgdfg

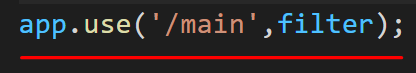
We can also use it to display message for some status like error and we are doing it by



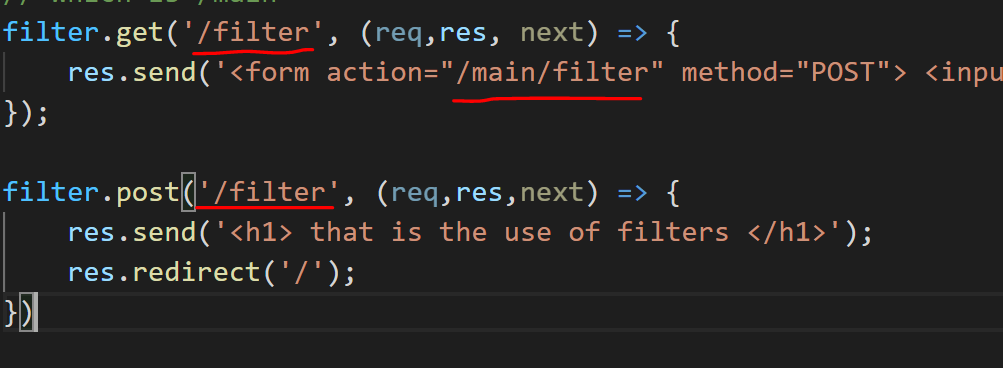
So we ca use these code at last to send h1 tag for the particular status obviously we have to use next or routs file before it so it can go down and check for every call best practice to calling by routs

* FILTER IN NODE JS

Filter is used to set up the common starting points so that instead of writing the /page address agin and agin we can write it a once.

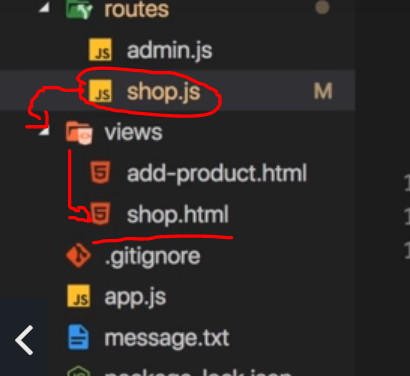


So that we can write export our filter without adding /main all the time we can write things ahed of it like /main/filter = /filter but we have to provide action as full address so that it can open that file /main/filter



* HOW TO SEND HTML FILES TO USER

So instead of writing html in res.send above we can write a html page separately

* So we can write it by using res.sendFile() instead of using res.send() and give the file path.
* But if we try to put a path that will give error because sendFile takes the path in form of linux system which is( /fff ) but window have a path with ( \fff )
* So o over came this error we have to require(‘path’) 
* And then used is as  path.join will join the path so that any user can use it.
* \_\_dirname in it is holding the absolute path of the folder so we can pass path from it
* But we have to give (.//) first because we are calling sendfile from some inside file so we are doing (.//) so that we can go to the root first and then give the path
* So we do not have to write (/) or (\) before any path
* The file from we are calling is shop.js
* 

If we try to add css file into our html page that file we are sending it will not send with it because we are sending a html file only but not the css in above code so for that we have to pass the static location of the folder so the html can use the css file in it



4 => working with dynamic content in node js

So how do we pass our input data from user to server for the request so we can pass it in the form of variable and array of object.

const products = [];

// /admin/add-product => POST

router.post('/add-product', (req, res, next) => {

  products.push({ title: req.body.title }); // push object inside array

  res.redirect('/');

});

exports.routes = router;

exports.products = products; // export it so it used by other files too

which is passing in the product page as admin data.product

const adminData = require('./admin');

const router = express.Router();

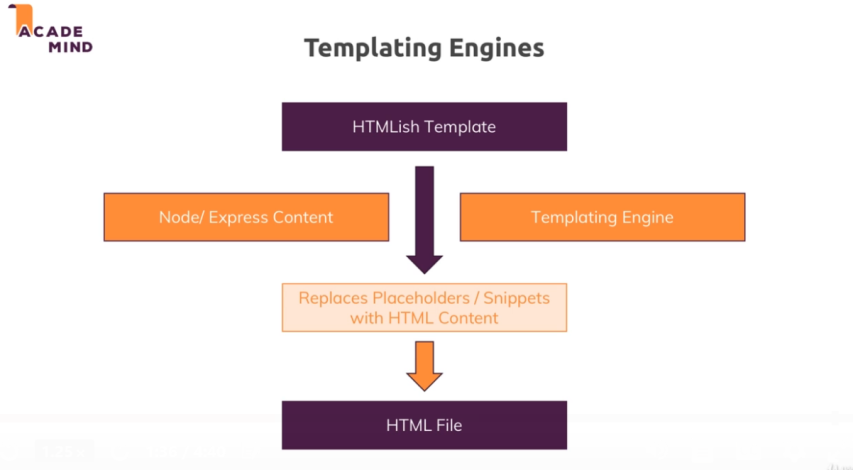
router.get('/', (req, res, next) => {

  console.log('shop.js', adminData.products);

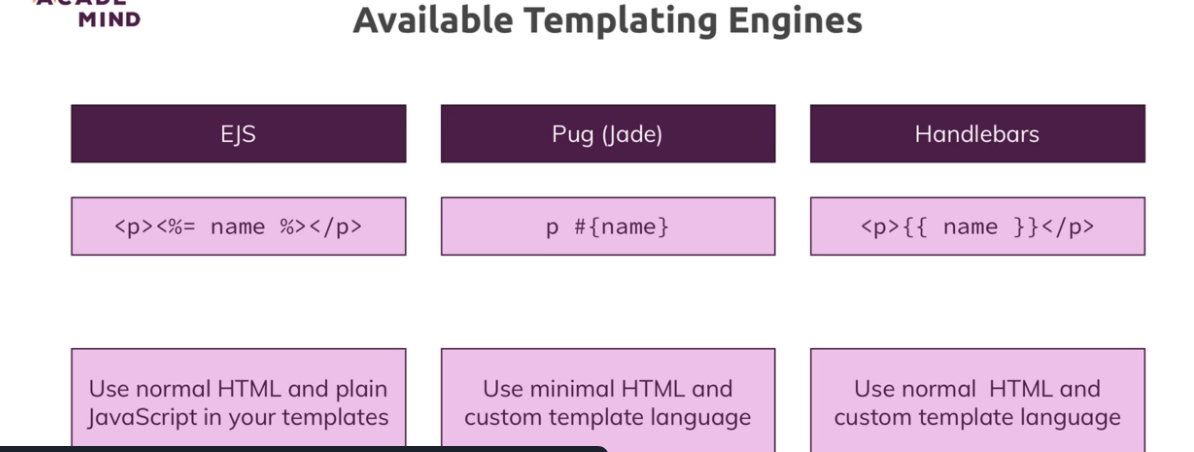
  res.sendFile(path.join(rootDir, 'views', 'shop.html'));

});

So to putting dynamic content to our html pages we uses templates to add templates we use templates engine so htmlish contains all the css,js and html with some place holders in between



so the templates engines are of different types some of them are and they use different syntax and you can add the dynamic content in it



So all the template can add dynamic content such as name in it but all have different syntax for that

* Installing templates

So we have to install these templates and we can do it by saving into dependencies



* Use of pug in place of html

So pug consider good in place of html if we are sending the file

To use pug first we have to define in our express that we also want to view the files with .pug extention

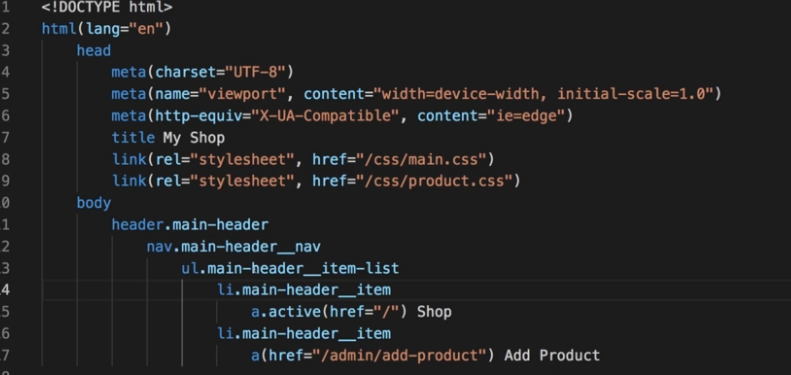
We are doing this so we can send the data in form of pug

We can do this by setting the global execution value so app.set will help us to set the value



So we are setting vie engine as pug and views as default

So the structure of pug file is



The class is defined by .---- and the links or things other the class and id will defined inside the ( curli breces ) and text at the end. So it work like html but in short line of code

Now to send it we have to give the path when ever we have to send and or this we have to render the file 

So it will render the file shop.html and get used in app as view engine

So we use pug to add some dynamic content and functionality as well so with pug we can call the data at runtime and use it

routs.get('/',(req,res,next) => {

    //res.send('<h1> send the data using Express </h1>');

    const products = product.product;

**res.render('shop',{prods: products, docTitle: 'shopPage'});**

    // now we can use product object b pods in our pug file as well as docTitle

});

Now we can add objects to the shop.pug file we can use the prods and docTitle and as we know product is an array of title page

So now our pug file look likes so we can add objects dynamically by using **#{inside it}**

doctype html

html(lang="en")

    head

        meta(charset="UTF-8")

        meta(http-equiv="X-UA-Compatible", content="IE=edge")

        meta(name="viewport", content="width=device-width, initial-scale=1.0")

        title **#{docTitle}**

    body

        header.head <!-- . add class to it -->

            ul

              li

                a.link(href="/product") go to link

        main

            if prods.length > 0

                .grid

                    each product in prods

                        article.card.product-item

                            header.card\_\_header

                                h1.product\_\_title **#{product.title}**

                            div.card\_\_image

                                img(src="data:image/jpeg;base64,/9j/4AAQSZJRgAB”, alt="A Book")

                            div.card\_\_content

                                h2.product\_\_price $19.99

                                p.product\_\_description A very interesting book about so many even more interesting things!

                            .card\_\_actions

                                button.btn Add to Cart

            else

                h1 No Products

we are using doctitle and productitle inside inside the pug file

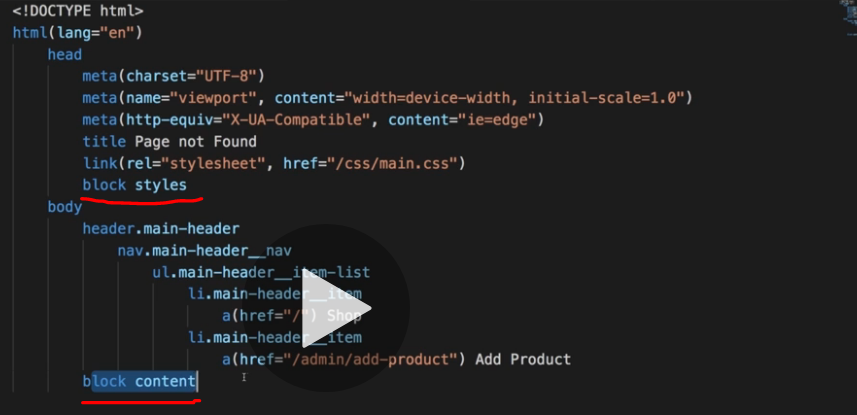
* Use of layout to store and optimize the repetitive code in a single folder

So we use layout to reduces the repetitive code, such as the html5 starting links,title,things before body and inside body two which is used repetitively in more then 1 file

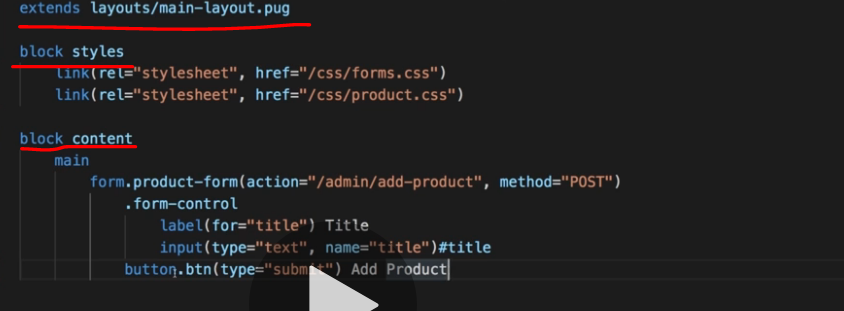
So if we have repetitive code using more then one time in more then 1 file so then we uses layout in it

So we can create a main layout pug file in it

But what happened if we wont to change a single line of code which is different for the different files. So we can name that line as **block** and name the line so that we can add something later in it



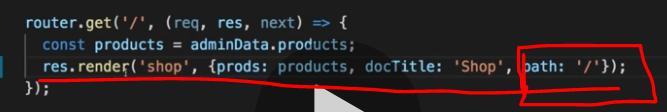
And then we can call this layout by **extends and the layoutfilepath** and fill the blocks if we want you can skeep all the blocks or any block that you want or not in use



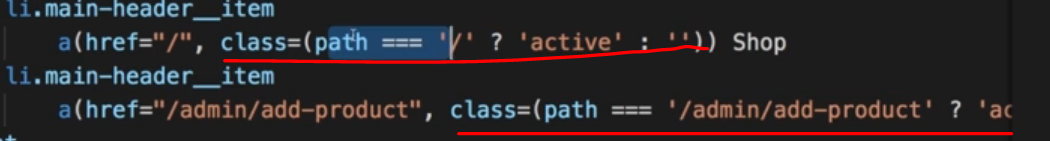
And if we add some class or id dynamically we have to use some js in the pug file

So we have to use js because what if all file have that same link but different stile and functionality for different page

So first we have to pass the path while render so that pug file can use it in run time



An after that for each different files we defines the path as different so that it can compare and check decide to add the class or not with adding class it also takes the css of it



* **EJS ENGINE**

We can use ejs instead of pug engine and the benefit of that is it uses normal html with the js in it.

Just like pug we have to define first that our engine is ejs and used in the folder

app.set(**'view engine','ejs'**);

app.set(**'views','views'**);

so we defined our view engine as ejs so we do not have to write the ejs extention while render and views as default folder to look the file at

so we are just passing the prods object containing the array and docTitle contains text

 res.render('shop',{prods: products, **docTitle: 'shopPage'**});

the render will only look to the set things in the app and go to the views folder

<title> **<%= docTitle %>** </title>

So we can add our data dynamically with the help of <%= inside this sing%>

**Difrent between <%= and <%- is that <%= takes the text formate while <%- takes includes the thing like path and <% takes javaScript code in it**

If some code is common in all then you can create a layout or copy the common code in somre different folder and used it by <%- includes(file path) %>

**<%- include('ejsLayouts/header.ejs')  %>**

Which is taken from the header.ejs

And also you can use common js code just like

**<% if (prods.length > 0) { %>**

            <div class="grid">

**<% for (let product of prods) { %>**

                    <article class="card product-item">

And close it when you have to by

               </article>

**<% } %>**

            </div>

**<% } else { %>**

            <h1>No Products Found!</h1>

**<% } %>**

So ejs having advantages over pug you can use the whole html code as it is with some java Script functionality on it.

* **MVC in node js**

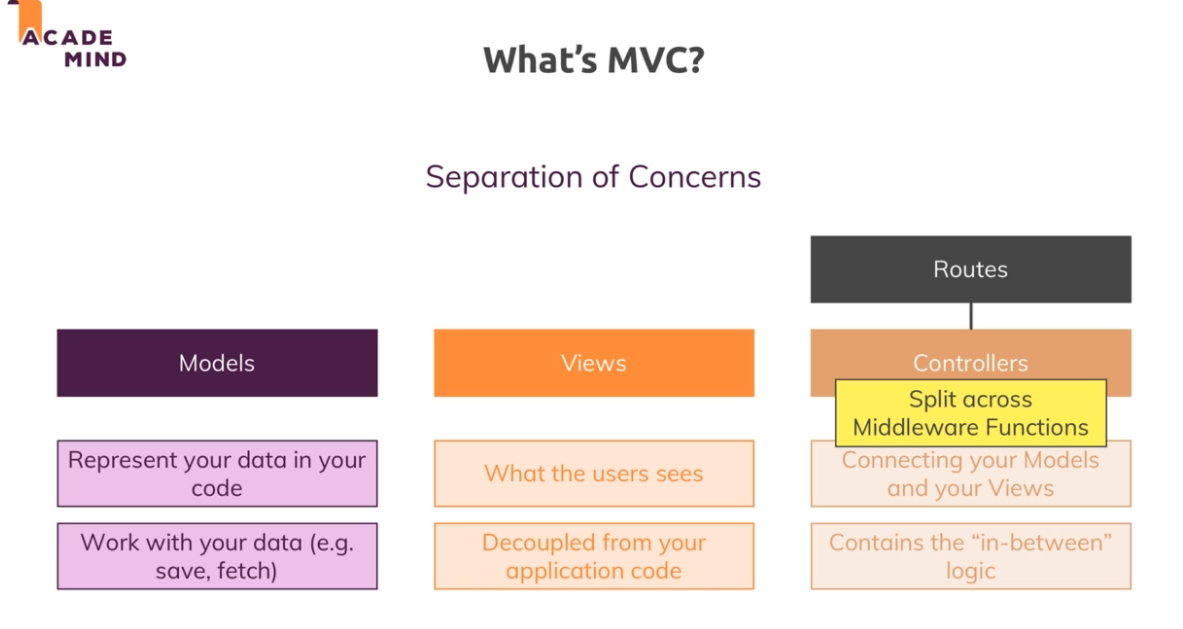
Mvc is used to separation of concern , by mvc you can separate your files and do know that which file is doing what

Mvc stands for model view controller

The model is responsible for representing the data in your code like save fetch even the code in memory

View id responsible for what the user see at the end responsible for rendering the write content into html uses templates

Controllers are the connection point between your models and views it contains the in between logic it controls which model should work with which view like routing the templates to user



So now we have to create our controller folder in which we can create our all the routes logic and export them so the routes of required them can call them

This code is in the controller file

exports.getAddProduct = (req,res,next) => {

    res.render('admin.ejs', { docTitle: 'productPage'});

}

// so you can export this function inside some routes

We are calling the above exported code in the adim.js file

// so that we can require file and can use exports in their

const getController = require('../controllers/products.js');

// here using getAddProduct export in the admin.get request

admin.get('/product',getController.getAddProduct);

and we can use all the routes code in one single file and can export them with different names so that different routes can use it

const addProduct = [];

exports.postProduct = (req,res,next) => {

    addProduct.push({title : req.body.title});

    res.redirect('/');

}

exports.getShop = (req,res,next) => {

    res.render('shop',{prods: addProduct, docTitle: 'shopPage'});

}

admin.post('/All-product',getController.postProduct);

routs.get('/',getController.getShop);

* **Modules in mvc**

Now we can use the modules to stores our data and save it and fetch it for this let us use the array of products in different space that we are using above

So for that facility we create a different file of product.js in our module folder which is used

To store thing in the array and use it when ever requires

// so modules plays with the data and the memory storage

const products = [];

module.exports = class product {

    constructor(t) {

        this.title = t;

    }

    save() {

        products.push(this); // push all the variables or function of the constructor

    }

    static fetch() {

        return products;

    }

}

So above we are exporting the class inside our controller file so that the use the product to save and fetch to the server when required

**const product = require('../models/product.js');**

exports.postProduct = (req,res,next) => {

**const products = new product(req.body.title);**

**products.save();**

    res.redirect('/');

}

exports.getShop = (req,res,next) => {

**const products = product.fetch();**

    res.render('shop',{prods: products, docTitle: 'shopPage'});

}

So instead of creating array inside the controller we are creating in different folder so that we manage the database at one specific file.

So above code we are getting the exports from the modules and saving the req title in product as product.save and then fetching it when required.

* ***Read and write file instead of pushing array of product in modulo***

So we know let us write file from server to in our data and read from our data to server

const { json } = require('body-parser');

const fs = require('fs');

const path = require('path');

const { setFlagsFromString } = require('v8');

module.exports = class product {

    constructor(t) {

        this.title = t;

    }

    save() {

**const p = path.join(\_\_dirname,'../','Data','product.json');**

**fs.readFile(p,(err,fileContent) => {**

**let product = [];**

**if(!err) { // check file present in the path or not**

**product = (JSON.parse(fileContent)); // parse because arry in form of array**

**}**

**product.push(this); push this content in array**

**fs.writeFile(p, JSON.stringify(product), (err) => { // write in the file**

**console.log(err);**

**})**

**})**

**}**

    static fetch() { // so to adjust the async we have to use a callback function in it which will run when it read the file

**fs.readFile(p, (err,fileContent) => { // fs.read is async function so to it will throw error if we access it before time**

**if(err) {**

**return [];**

**}**

**return JSON.parse(fileContent);**

**})**

**return products;**

**}**

}

So we have to pass the array all time and because we are storing into some file that’s why we have to stringify it because we are writing the data and to readFile we have to parse it

To save file we have to check for error and that is if file is present in the file if not then we write the empty array

**But if we return something inside the read file it will throw an error because the return value get run before the file reads completely because read file is a async function so for that we have call a function which can hold the return values and it will run only when the async read run completely.**

**static fetch(cb) { // so to adjust the async we have to use a callback function in it which will run when it read the file**

**const p = path.join(\_\_dirname,'../','Data','product.json');**

**fs.readFile(p, (err,fileContent) => { // fs.read is async function so to it will throw error if we access it before time**

**if(err) {**

**cb([]); // we are using function so it will run only once the read file runs**

**}**

**cb(JSON.parse(fileContent)); // read file is async so it will run when it reads the whole**

**})**

**return products;**

**}**

**}**

**So here we are calling cb as call back function which run when the async function comes in the call back stack**

**exports.getShop = (req,res,next) => {**

**product.fetch((products) => {**

**res.render('shop',{prods: products, docTitle: 'shopPage'});**

**});**

**So we can call fetch as like above in which we are calling a product which now holding an array**

**So now we add all the necessary files which is required to create product page so we just bit modified our pages**

**Dynamic Routes & Advanced Models**

Now we have to pass some data anusing them dynamically such as the product id so we can call that thing by their id or send by id or sending the data using url’s

So to add an id you have to add id by using Math.random() which will give random number

You have to pass id with this.id in the product class during the saving time

 save() {

**this.id = Math.random().toString();**

    getProductsFromFile(products => {

      products.push(this);

  <div class="card\_\_actions">

**<a class="btn" href="/products/<%= product.id  %>">Details</a>**

Then you have to pass the link in detail link so it opens the page with product id

* Dynamic routs in node.

Dynamic routs represented by /:someName it is used to save the the path after (/) in the :some name

Suppose you are passing some **id (/product/4586**) then :productId will take that 4586

router.get('/products**/:productId'**, shopController.getProduct);

so it will take any thing if you pass after /product/anything but notice if you pass /delete after it it will note open the delete file if you have any in case it will saves it as Dynamic

**if you have as such file present make shore you write it before and at last write the dynamic /:product because if you do that then only file read the delete other wise it will read /:product first because rout move top to bottom**

router.get('/products/delete', shopController.getProduct);

router.get('/products/:productId', shopController.getProduct);

and we can extract these productId in shopController.getproduct by use of params which can take the params of the path

exports.getProduct = (req,res,next) => {

**const proId =  req.params.productId;**

  console.log(proId);

  res.redirect('/');

}

So if we want to get particular element by id so we have to create that element in our product class in our model

So we create a function which takes the id and return the call back function containing particular element.

 static findById(id,cb) { // cb is call back which will get by the method which calling it

    getProductsFromFile(products => { // products is array getting from this function above

      const product = products.find(p => p.id === id); // find while find the particular element from array

      cb(product);

    });

  }

So we call cb and passing our particular product with that we are getting product as call back which we are console.log in screen

exports.getProduct = (req,res,next) => {

  const proId =  req.params.productId; // it stores the product id getting by /product/:productId

  Product.findById(proId, product => {

    console.log(product);

  })

  res.redirect('/');

}

So now we create our product detail page and pass a product array while render

exports.getProduct = (req,res,next) => {

  const proId =  req.params.productId; // it stores the product id getting by /product/:productId

  Product.findById(proId, product => {

    res.render('shop/product-detail.ejs',{

      product: product,

      pageTitle: product.title,

      path: '/product'

    });

  });

Now we create a form button add to cart so when ever we press we redirect to the cart page and latter on we can add the detail of the product init

<form action="/cart" method="POST"> <!-- we can pass the data using post request-->

    <button class="btn" type="submit">Add to cart</button>

    <input type="hidden" name="productId" value="<%= product.id %>">

    <!-- it will pass the value by post request in method-->

</form>

So we are using method post so the value in input can bet get by post method In the router by the key as name (productId)

exports.postCart = (req,res,next) => {

  // we get it from input value inside form method post

  const prodId = req.body.productId; //we can get the value during method post

  console.log(prodId);

  res.redirect('/cart');

}

router.post('/cart', shopController**.postCart**);

now we have to store the data of cart which we are going to save in **cart.json inside model**

**here we check the product if it is present then update the quantity only but if not present then update the cart by adding new product in** it

// we have to collect and save the cart detail so we hav e to create a model for that

const fs = require('fs');

const path = require('path');

const p = path.join( // file path where we want to read or write

    path.dirname(process.mainModule.filename),

    'data',

    'cart.json'

);

module.exports = class cart {

    static addProduct(id,productPrice) {

        //fetch the previous product

        fs.readFile(p, (err,fileContent) => {

            // creating cart so that can pass it in form of json

            let cart = {products: [], totalPrice: 0}

            if(!err) {

                cart = JSON.parse(fileContent); // add previous file content

            }

            // find the index of that id so if found we can update the quantity if not then update the json file

            const existingProductIndex = cart.products.findIndex(p => p.id === id);

            const existingProduct = cart.products[existingProductIndex];

            let updatedProduct;

            if(existingProduct) { // if not null then increase the quantity by one

               updatedProduct = {...existingProduct }

               updatedProduct.qty += 1;

               cart.products = {...cart.products}

               cart.products[existingProductIndex] = updatedProduct;

            }

            else { // if their is no product such like that present in data then update the data

                updatedProduct = {id: id, qty: 1}

                cart.products = [...cart.products , updatedProduct];

            }

            cart.totalPrice = cart.totalPrice + +productPrice; // increase the price in form of integer by + +

            fs.writeFile(p,JSON.stringify(cart), (err) => {

                console.log(err);

            })

        })

    }

}

Which we calling this from our shop controller from shop.js inside controller folder

exports.postCart = (req,res,next) => {

  // we get it from input value inside form method post

  const prodId = req.body.productId; //we can get the value during method post

  Product.findById(prodId, (product) => {

    cart.addProduct(prodId, product.price); //it will run once we get product from callBack

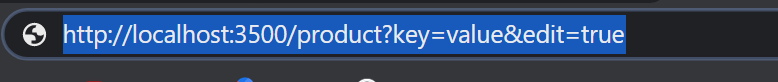
  })

  res.redirect('/cart');

}

* **Query Parameters in node**

Query parameters are the key value pair which can be used after ? in our url



<http://localhost:3500/product?key=value&edit=true>

here we can add key value using dynamic content in url by example => /:product

an we can pass it in our ejs file so we pass some key value pair also which can be used when we need some dynamic data



if admin wants to edit the page the admin press edit button so after pressing the edit button admin should get the old product detailed filled up on the page

and then he can edit the data and when press update button the data should be update on the memeory can do it by get nd post the data by admin.js

// when admin wants to edit/update the product admin will get the detail of that product in the page

exports.getEditProduct = (req, res, next) => {

  const editMode = req.query.edit;

  if (!editMode) {

    return res.redirect('/');

  }

  const prodId = req.params.productId;

  Product.findById(prodId, product => {

    if (!product) {

      return res.redirect('/');

    }

    res.render('admin/edit-product', {

      pageTitle: 'Edit Product',

      path: '/admin/edit-product',

      editing: editMode,

      product: product

    });

  });

};

// when admin change something from get detail content it will save in same index with update

exports.postEditProduct = (req, res, next) => {

  const prodId = req.body.productId; // id get from edit-product.ejs from hidden input

  const updatedTitle = req.body.title;

  const updatedPrice = req.body.price;

  const updatedImageUrl = req.body.imageUrl;

  const updatedDesc = req.body.description;

  const updatedProduct = new Product(

    prodId,

    updatedTitle,

    updatedImageUrl,

    updatedDesc,

    updatedPrice

  );

  updatedProduct.save(); // update the the new data of product

  res.redirect('/admin/products');

};

So to save file then data check for the product is updated or the product is new one so in our save function if we the id is null that means the data is new otherwise data is updated

save() { // here it is check the product is new or you are editing the previous

    getProductsFromFile(products => {

      if (this.id) { // if id present that means the product is old one and you want to edit

        const existingProductIndex = products.findIndex(

          prod => prod.id === this.id

        );

        const updatedProducts = [...products];

        updatedProducts[existingProductIndex] = this; // passing updated this value in th updated index

        fs.writeFile(p, JSON.stringify(updatedProducts), err => {

          console.log(err);

        });

      } else { // if the id is null therefore creating the random id

        this.id = Math.random().toString();

        products.push(this);

        fs.writeFile(p, JSON.stringify(products), err => {

          console.log(err);

        });

      }

    });

Now we have to show our items in cart with the help of cart data we can add the element in our cart but first we have to create our cart page

  <body>

        <%- include('../includes/navigation.ejs') %>

        <main>

            <% if (products.length > 0) { %>

                <ul class="cart\_\_item-list">

                    <% products.forEach(p => { %>

                        <li class="cart\_\_item">

                            <h1><%= p.productData.title %></h1>

                            <h2>Quantity: <%= p.qty %></h2>

                            <form action="/cart-delete-item" method="POST">

                                <input type="hidden" value="<%= p.productData.id %>" name="productId">

                                <button class="btn danger" type="submit">Delete</button>

                            </form>

                        </li>

                    <% }) %>

                </ul>

            <% } else { %>

                <h1>No Products in Cart!</h1>

            <% } %>

In the cart page we run for loop in the product array which is present in cart.JSON and we get our data from there and and display inside for loop

And to show the data in our ejs file we have to pass our data by the use of model to controller so that controller can send that data to the view

// it is used to pass the cart.json data into shop.js controller

  static getCart(cb) {

    fs.readFile(p, (err, fileContent) => {

      const cart = JSON.parse(fileContent);

      if (err) {

        cb(null); // means cart.json is empty

      } else {

        cb(cart);

      }

    });

  }

Now we call that get cart in our controller shop.js

exports.getCart = (req, res, next) => {

  Cart.getCart(cart => {

    Product.fetchAll(products => { // fetching the particular product to cart

      const cartProducts = [];

      for (product of products) {

        const cartProductData = cart.products.find(

          prod => prod.id === product.id

        );

        if (cartProductData) {

          cartProducts.push({ productData: product, qty: cartProductData.qty });

        }

      }

      res.render(**'shop/cart'**, {

        path: '/cart',

        pageTitle: 'Your Cart',

        products: cartProducts // contains the cart data which will shown in cart.ejs page

      });

    });

  });

};

So the data will render throw **shop/cart.ejs**