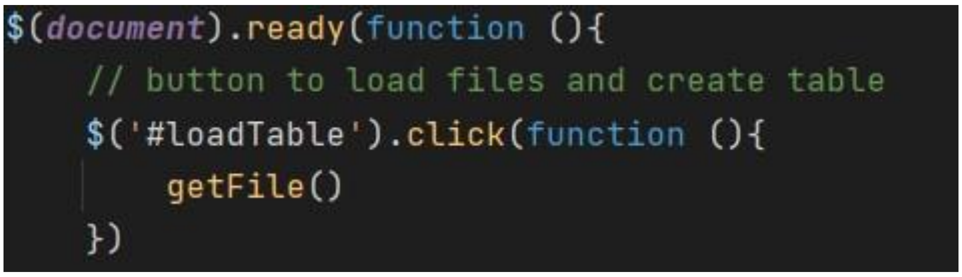
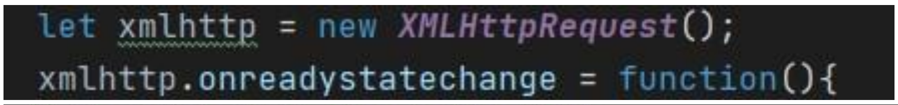
## Q1

(a) AJAX can facilitate the retrieval of data, and update the page, without having to refresh the page. The browser acts as the client, sending requests to the server, expecting a response to display to the user (or use for further processes to display to the user). So the page loads up, then lets say the user clicks a button. The listnener of that button will make a function,telling The browser to make a Ajax XMLHttprequest to the server as

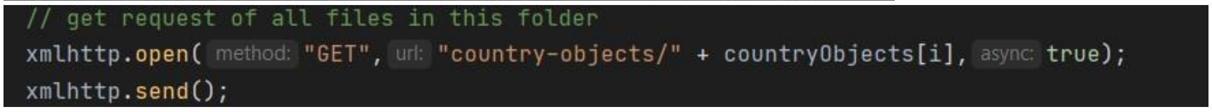


```
$(document).ready(function (){
    // button to load files and create table
    $('#loadTable').click(function (){
        getFile()
    })
```

This is using Jquery, when the button with id load table gets called, the function getFile is called



```
let xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function(){
```



```
// get request of all files in this folder
xmlhttp.open( method: "GET", url: "country-objects/" + countryObjects[i], async: true);
xmlhttp.send();
```

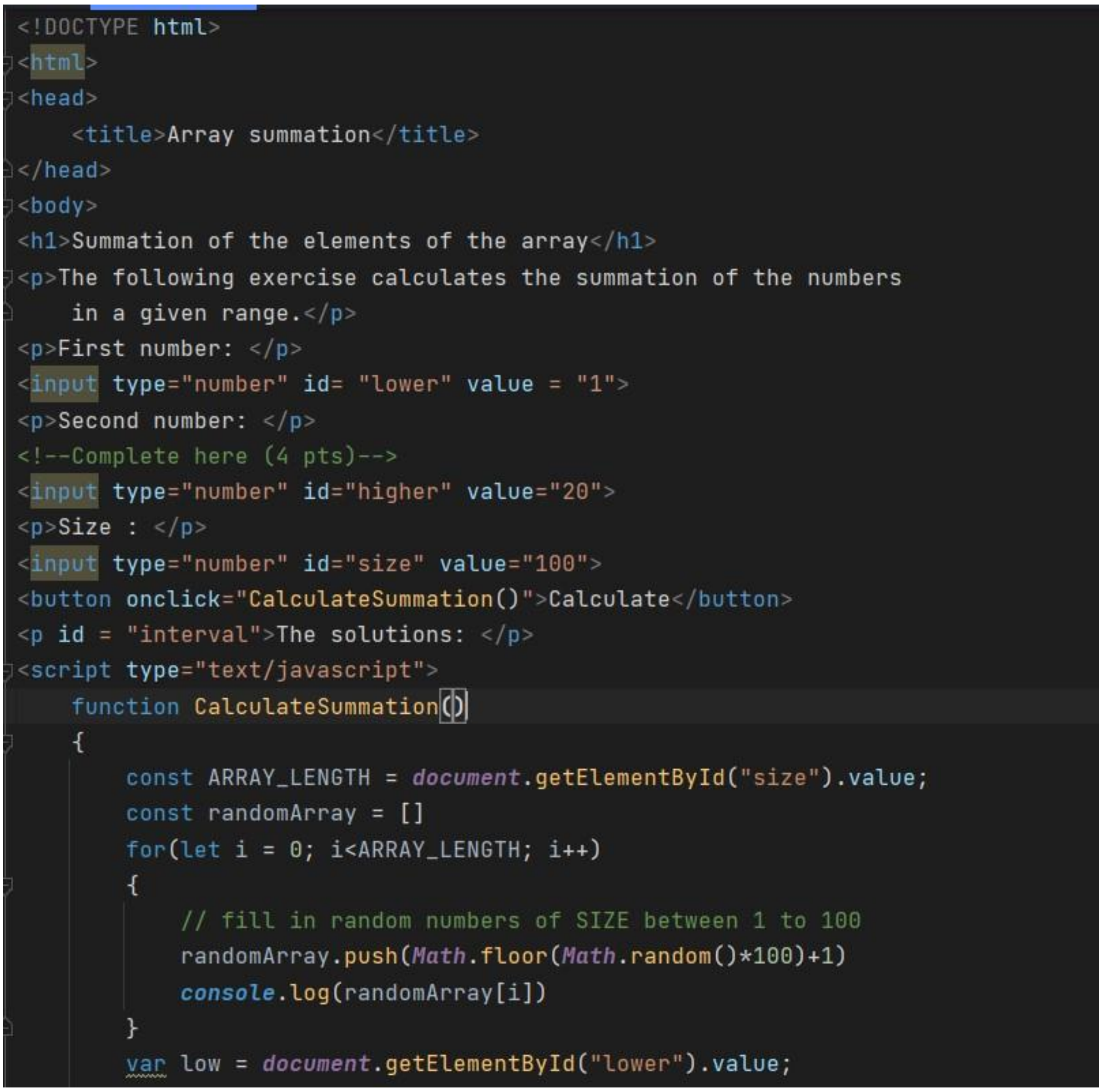
Getfile creates a GET XMLHttprequest to retrieve the resource “country-objects” in which upon receiving the request, the server will obtain the resource (and do other functions and processes) then send back the required resource back to the client, in which will be rendered in the HTML page, without refresh.

So in summary, if an event occurs, a XMLHTTPrequest is sent to the server, in which the server processes this and sends a response back to the client, in which is read by JavaScript and rendered to the page without refreshing.

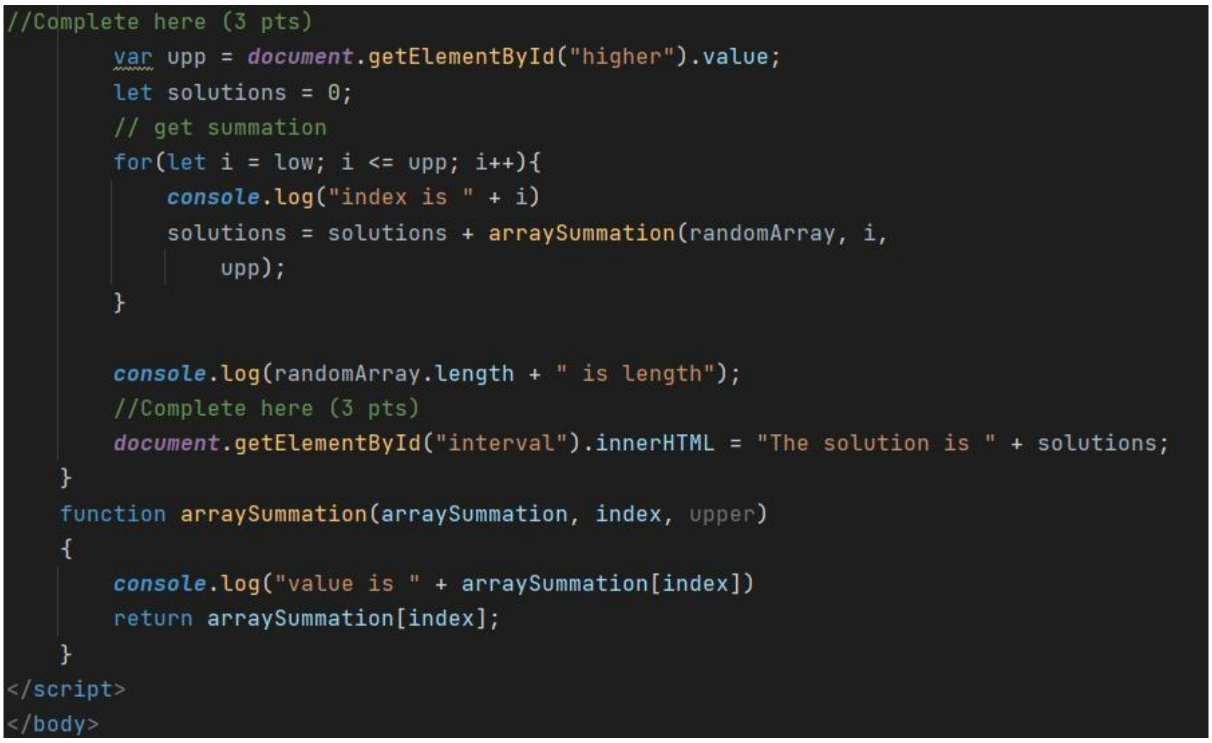
(b) REST is a protocol defining how resources travel using HTTP protocols. Other APIs used could be SOAP, which offers more authorization and stricter and more advanced security features such as ACID compliance. However, it comes with the cost of being slower in terms of loading pages, more demanding as it is more complex and consumes far more bandwidth. When it comes to data, enterprise should consider whether better performance or security is more important. Applications such as social apps or games, should focus more on performance for user satisfaction however applications where security is of upmost importance such as bank applications or forensics software should be using SOAP instead of REST.

In regard to this company, being hired for, information must be kept confidential, hence SOAP must be taken into account more, SOAP uses only XML as it’s messaging format however as stated has high security. To implement SOAP Is more function driven, the data is in the form of services rather than just data to retrieve, meaning more processing as more functions are being used, resulting in more complex code and processing power. SOAP does offer a variety of protocols when transferring web traffic such as TCP, SMTP and UDP unlike REST. REST is a lightweight protocols, so if wanting to use REST, which is more flexible, has multiple response formats, can be used with multiple applications and languages, however lacks security, however it can be added by building on top of HTTP, however doesn’t come with it as part of the package.

(c) Generate an array of the size, given by user and fill it with random values from 1 – 100



```
<!DOCTYPE html>
<html>
<head>
    <title>Array summation</title>
</head>
<body>
<h1>Summation of the elements of the array</h1>
<p>The following exercise calculates the summation of the numbers
    in a given range.</p>
<p>First number: </p>
<input type="number" id= "lower" value = "1">
<p>Second number: </p>
<!--Complete here (4 pts)-->
<input type="number" id="higher" value="20">
<p>Size : </p>
<input type="number" id="size" value="100">
<button onclick="CalculateSummation()">Calculate</button>
<p id = "interval">The solutions: </p>
<script type="text/javascript">
    function CalculateSummation()
    {
        const ARRAY_LENGTH = document.getElementById("size").value;
        const randomArray = []
        for(let i = 0; i<ARRAY_LENGTH; i++)
        {
            // fill in random numbers of SIZE between 1 to 100
            randomArray.push(Math.floor(Math.random()*100)+1)
            console.log(randomArray[i])
        }
        var low = document.getElementById("lower").value;
```



```
//Complete here (3 pts)
        var upp = document.getElementById("higher").value;
        let solutions = 0;
        // get summation
        for(let i = low; i <= upp; i++){
            console.log("index is " + i)
            solutions = solutions + arraySummation(randomArray, i,
                upp);
        }

        console.log(randomArray.length + " is length");
        //Complete here (3 pts)
        document.getElementById("interval").innerHTML = "The solution is " + solutions;
    }
    function arraySummation(arraySummation, index, upper)
    {
        console.log("value is " + arraySummation[index])
        return arraySummation[index];
    }
</script>
</body>
```

Q2

(a) Given the companies concern for high level of transactions, I believe a NoSQL database like MongoDB would be a far more suitable database to use than SQL based databases. The main reason of this is that there will be a high level of transactions, in which relational databases were not designed to be able to cope and adapt to the scale of data, and is unable to take full advantage of the amount of storage and processing power. Imagine having a table keeping track of a User. Soon all users can stored in 1 table, making a huge block of data. Soon certain fields may need to be added in for certain users, hence it will have to update for all users in case, which may lead to certain users having empty fields, e.g Mary has a middle name, entered in database, currently all previous users have no surname, so left empty. It is also difficult to embed data (maybe one user has 2 addresses). It will need to be split into 2 separate tables and referenced instead (so 1 user has 2 references in address table).

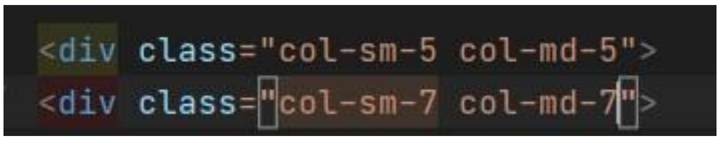
MongoDB allows for embedded data, and all it's fields can be different from each other (one user can have middle name, another cannot). As well as that, the company is using NodeJs, assuming it's a JavaScript application, MongoDB uses JavaScript, it's documents are in the form of JavaScript objects, making it much easier to handle the data, as using same language. In terms of large data, MongoDB would be more useful as it uses Sharding instead of portioning. Sharding means the application is connected to several different servers, rather than just one (partitioning). Data is stored across different servers, in which a router can be used as it contains a mapping of which shard as what data, allowing for it to make queries to the appropriate database server containing the data required.

(b) Bootstrap can provide for responsive user interface design.

Users in this era of technology use applications in several different technologies in all different sizes, such as mobiles, tablets, laptops and monitors. In this case, if an enterprise would consider deploying their application to a mobile device, then they should take into consideration offering a responsive interface, hence here is where Bootstrap comes in to play. The benefit of responsive interface is that the interface becomes not only more stylish and appealing, but more importantly more user-friendly. Without responsive interfaces there may be buttons out of place, tabs and writing going out of bounds and a lot more mess, giving the user a lot more trouble using the application, and eventually would get fed up with it. A lot of consideration needs to be taken as screen sizes affect a lot of aspects of user actions, such as visual and readability, typing space. Mobile users generally would only use web applications on mobile for short actions such as a quick search, hence needs to be designed to accommodate quick and few actions in order to satisfy the user request.

An example of how Bootstrap can facilitate responsive design, is it's grid system.

The bootstrap grid system provides built in classes to allow for easy use to split the web page to sized columns and rows. It allows for easy use to adjust the size of the column sections. Since for a column section can be added up to 12 for a row, it allows web developers to easily visualise how to split up certain sections such as text or navbars into sections allowing for easy use of making columns and rows. On top of that one of the advantages when it comes to responsive design is the ability to add in multiple scales within the same class such as the example below



```
<div class="col-sm-5 col-md-5">
<div class="col-sm-7 col-md-7">
```

As can be seen, here facilitates multiple sizes, such as small devices being 5/12 – 7/12 split, but also catering for medium size devices. With this, bootstrap's grid system can easily satisfy responsive design, as it caters to a number of different screen sizes. Bootstrap provides a lot of built in classes useful for mobile, such as modals or pop ups (as screens are small, sometimes pop ups are necessary). It provides toast, which is the temporary alert for example when you set a timer It is to note that developers would rarely stick to one platform, for example web applications these days would not only make their application only available on browsers from laptops. They would make it available on as may devices and screens possible, hence the need for responsive design, explaining the growth of bootstrap's popularity of styling with the increase of sales in mobile phones.

( c ) This can be implemented using the REST API. The principles of REST is of uniform interface, to separate and define the roles of client and server so they act independently. Resources can be retrieved through identifiers called URLs, and can be modified if needed. Using the REST API, to retrieve the gallery items to be rendered in the page, the gallery items would be the data resources, and should be identified by different URLs of the page. It is to be noted that the REST API must have a uniform interface, all resources (gallery items) should be different from the representation that are returned to the client (so it would be returned as JSON objects containing the name, description of the item).

The URLs chosen should be named as simple and obvious as possible, users should understand just by looking at them. Assuming they have 2 items "pictures" and "sculptures", then possible URLs for these resources could be http://www.exihibition.com/year-2020/museums/gerogria-musem/items/pictures http://www.exhibition.com/year-2019/musems/henry-museum/items/sculptures

As you search deeper, the resource becomes more specific, as can be seen from year to specific piece of artwork

So there are multiple types of requests that can be used to manipulate and retrieve the data , such as updating and deleting. These 4 types of requests or CRUD operations are GET,POST,DELETE and PUT.

So assuming the user would like to go a specific picture of an apple for an example. By clicking the link to go to that page, the browser will actually make a HTTP GET request, in the form of GET http://www.exihibition.com/year-2020/museums/gerogria-musem/items/pictures/apple HTTP/1.1, to the host http://www.exhibition.com. This request will be sent to the serve in which, the server can send back a JSON response {pictures : apple}, and a status code 200 if succeeded, or 404/500 if errors occurred.

A similar structure applies to the rest, client sends the request to server host, server responds with status and sometimes new URLs if newly created resources are requested (POST).

Should The museum would like to add a new item, for example a picture of a pear into the museum , it can add a new course, using the POST request. POST http://www.exihibition.com/year2020/museums/gerogria-musem/items/pictures HTTP/1.1 would be used to create a new picutre, passing in the details of the picture to the server such as id, name, whether through a HTML form or an Ajax/axios post request , in which the server can return to the client the new URL http://www.exihibition.com/year-2020/museums/gerogria-musem/items/pictures/pear

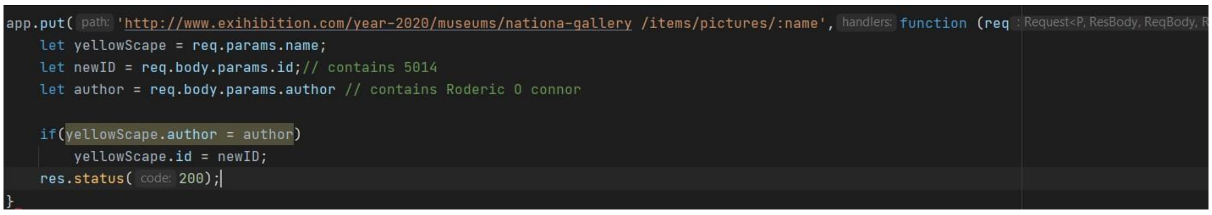
PUT requests can be used in order to edit a item, whether it's name or color etc.. if pear's author wad actually Joe instead of Mary, The museum may want to change it's author to Joe. For this it can make a PUT request for the specific URL it wants to edit. PUT http://www.exihibition.com/year2020/museums/gerogria-musem/items/pictures/pear HTTP 1.1, passing in the entire new representation of the object (with the updated author), in which the server will process it, and return a response code informing the client if it was a success (status code 200) or a failure (status code 404).

Finally Delete requests can remove a certain resource. Similar to PUT, the specific URL would need to be passed in to the server, however the full representation of the object is not needed, only it's URL as it's identification. So DELETE http://www.exihibition.com/year-2020/museums/gerogriamusem/items/pictures/pear, would remove the pear from the website, when showing what pictures are available.

(b)An example of retrieving a specific item from the website would be to use the GET request and to pass in the specific URL to the server, so for example GET http://www.exihibition.com/year2020/museums/gerogria-musem/items/pictures, would retrieve all pictures from the year 2020 from the Georgia museum. However, to retrieve all the items within the year 2020 would be to pass in the URL to items, so GET http://www.exihibition.com/year-2020/museums/national museum/items

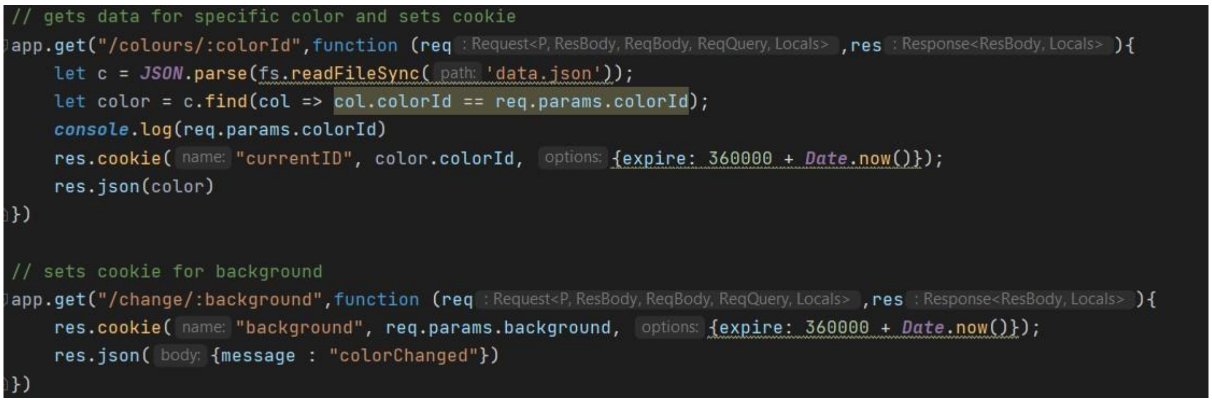
would retrieve a list of items within the year 2020.

© PUT http://www.exihibition.com/year-2020/museums/nationa-gallery /items/pictures/yellowlandscape



```
app.put( path: 'http://www.exihibition.com/year-2020/museums/nationa-gallery /items/pictures/:name', handlers: function (req : Request<P, ResBody, ReqBody, R
    let yellowScape = req.params.name;
    let newID = req.body.params.id;// contains 5014
    let author = req.body.params.author // contains Roderic O connor

    if(yellowScape.author = author)
        yellowScape.id = newID;
    res.status( code: 200);
}
```

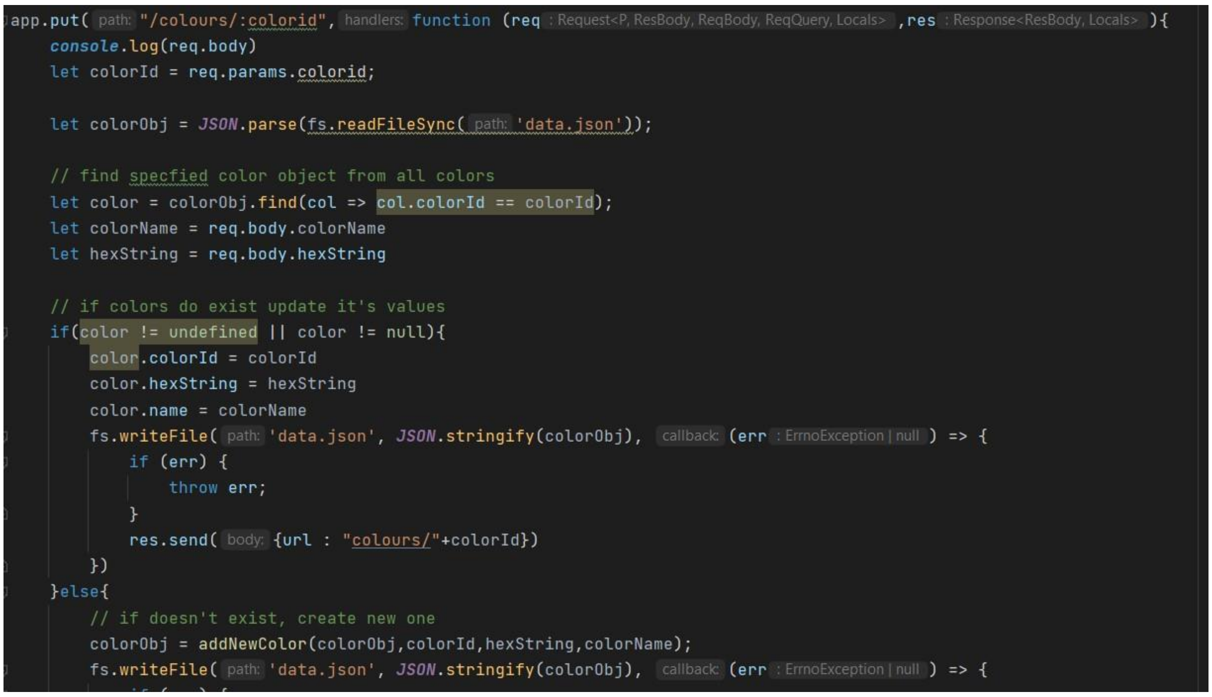
3 (a)  Routing can be described as how the application responds based on receiving a client request, specific HTTP request (GET or DELETE) or URL specified. Different routes will handle requests differently, the same route can handle requests differently as well based on the HTTP method sent. For example, in express here are 2 different routes



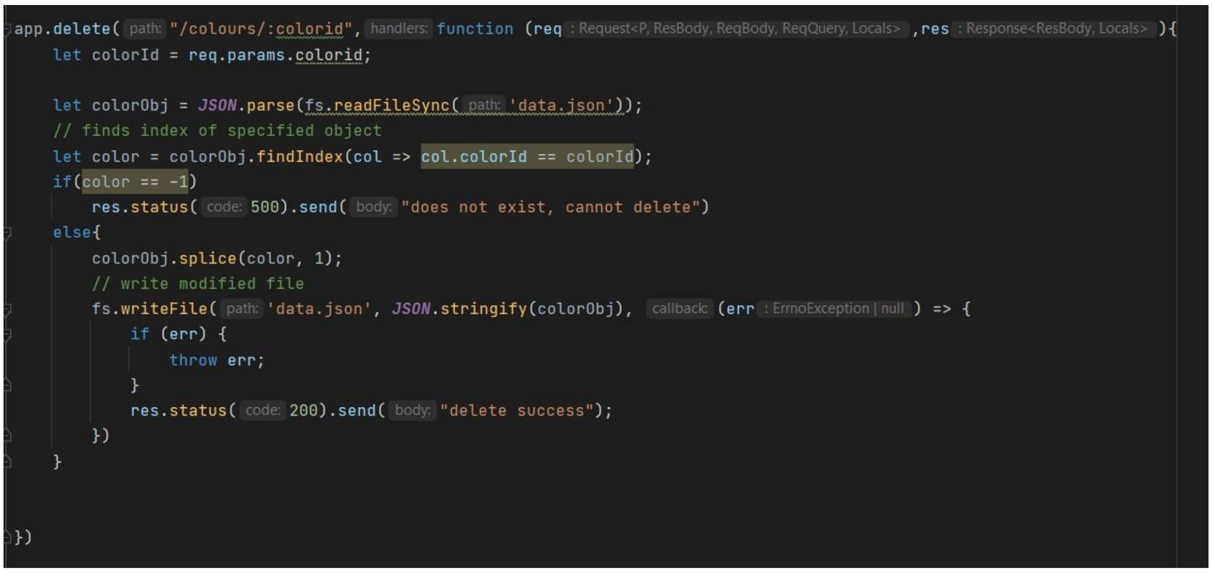
```
// gets data for specific color and sets cookie
app.get("/colours/:colorId",function (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> ,res : Response<ResBody, Locals> ){
    let c = JSON.parse(fs.readFileSync( path: 'data.json'));
    let color = c.find(col => col.colorId == req.params.colorId);
    console.log(req.params.colorId)
    res.cookie( name: "currentID", color.colorId,  options: {expire: 360000 + Date.now()});
    res.json(color)
})

// sets cookie for background
app.get("/change/:background",function (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> ,res : Response<ResBody, Locals> ){
    res.cookie( name: "background", req.params.background,  options: {expire: 360000 + Date.now()});
    res.json( body: {message : "colorChanged"})
})
```

Both are using GET, but as can be seen it’s 2 different URLs, hence the application responds differently to each (one gets a specific color, and one sets a cookie).

Here are 2 same URLs with different HTTP methods



```
app.put( path: "/colours/:colorid", handlers: function (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> ,res : Response<ResBody, Locals> ){
    console.log(req.body)
    let colorId = req.params.colorid;

    let colorObj = JSON.parse(fs.readFileSync( path: 'data.json'));

    // find specfied color object from all colors
    let color = colorObj.find(col => col.colorId == colorId);
    let colorName = req.body.colorName
    let hexString = req.body.hexString

    // if colors do exist update it's values
    if(color != undefined || color != null){
        color.colorId = colorId
        color.hexString = hexString
        color.name = colorName
        fs.writeFile( path: 'data.json', JSON.stringify(colorObj),  callback: (err : ErrnoException | null ) => {
            if (err) {
                throw err;
            }
            res.send( body: {url : "colours/"+colorId})
        })
    }else{
        // if doesn't exist, create new one
        colorObj = addNewColor(colorObj,colorId,hexString,colorName);
        fs.writeFile( path: 'data.json', JSON.stringify(colorObj),  callback: (err : ErrnoException | null ) => {
```

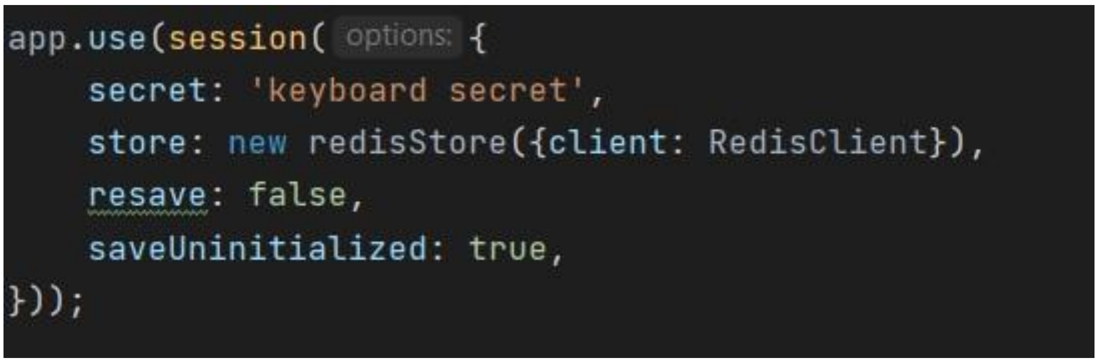


```
app.delete( path: "/colours/:colorid", handlers: function (req : Request<P, ResBody, ReqBody, ReqQuery, Locals> ,res : Response<ResBody, Locals> ){
    let colorId = req.params.colorid;

    let colorObj = JSON.parse(fs.readFileSync( path: 'data.json'));
    // finds index of specified object
    let color = colorObj.findIndex(col => col.colorId == colorId);
    if(color == -1)
        res.status( code: 500).send( body: "does not exist, cannot delete")
    else{
        colorObj.splice(color, 1);
        // write modified file
        fs.writeFile( path: 'data.json', JSON.stringify(colorObj), callback: (err : ErrnoException | null ) => {
            if (err) {
                throw err;
            }
            res.status( code: 200).send( body: "delete success");
        })
    }


})
```

One is a PUT method, and one is a DELETE method, one edits a color, the other deletes one. As can be seen with these requests, route parameters can also be assigned, which represents a value in the actual URL, for example => user/:id, where these route parameters are captured in the req.params object , so req.params { id : 5}, in the actual URL would be represented as user/5. This is a way to access any user, the route parameter can be passed in making it more generalised.

(c) Authentication is trying to authenticate users by validating them. It is to make sure any user is who they claim to be. An example of authentication would be a user logging in, to make sure the user is who they say they are, they have to not only provide the user name, but also provide their password. By writing in the password matching to the user, it is an act of authentication, as this user has provided some proof that they are who they say they are by knowing the password of the user. Another example of authentication, in terms of enterprise security would be. Authorization however is making sure a specific user has access to resources that they have permission or access to. An example would be an administrator would have access to certain resources and data that would be forbidden and confidential against a normal user. In a business setting, authentication would be swiping your ID card to get into the building, but authorization would be a higher level authority rank like Manager going into restricted places of the building. So works the same way in an application level, all staff would need to log in to their staff accounts, this is authentication, however accountants may be able to access financial data, in which developers can't, or security can access footage when cafeteria staff can't, this is authorization, certain users cant access certain things, passed on permission, and it must be assured users who access data have permission to do so, while authentication verifies that user's are wo they are.

(d) When for example the user logs in , the server can then create a cookie, storing, the user name and expiry date of the session (automatically destroyed If the user logs out). It can create the session and store using technologies such as a Redis server



```
app.use(session( options: {
    secret: 'keyboard secret',
    store: new redisStore({client: RedisClient}),
    resave: false,
    saveUninitialized: true,
}));
```

From that, sessions are stateful, hence the user data is stored in the server memory, after the server creates a cookie and sends it back to the browser, with the session ID in it. The cookie and session has been created, so next time , for example the user makes a request to go to the page to log in, since the cookie and session is not expired, when making a request, the cookie is also sent with the request. The server will examine the session ID, and retrieve the specific user and hence they get logged in immediately.

JWT however, being stateless, when the user for example sends a POST request to login with their username and password, the user data does not get stored in the server memory, instead a JWT will get created, with a secret encoded key made from an algorithm (such as HMAC). Note that the user information is stored in the JWT itself, still making it stateless as client server requests are independent and the server will not recognise the user as it retains no memory of it (without the JWT token). From that, the server will respond to the browser, passing in the JWT with it. The browser can store the JWT, in for example cookies. Next time the client makes a request, it will send the JWT with it, in which all the server needs to do is verify that the JWT signature is valid, and from that, it will get the user information that's within the JWT (as data is not stored in the server unlike cookies)
They work similar, main difference being is that one is stateful and one is stateless