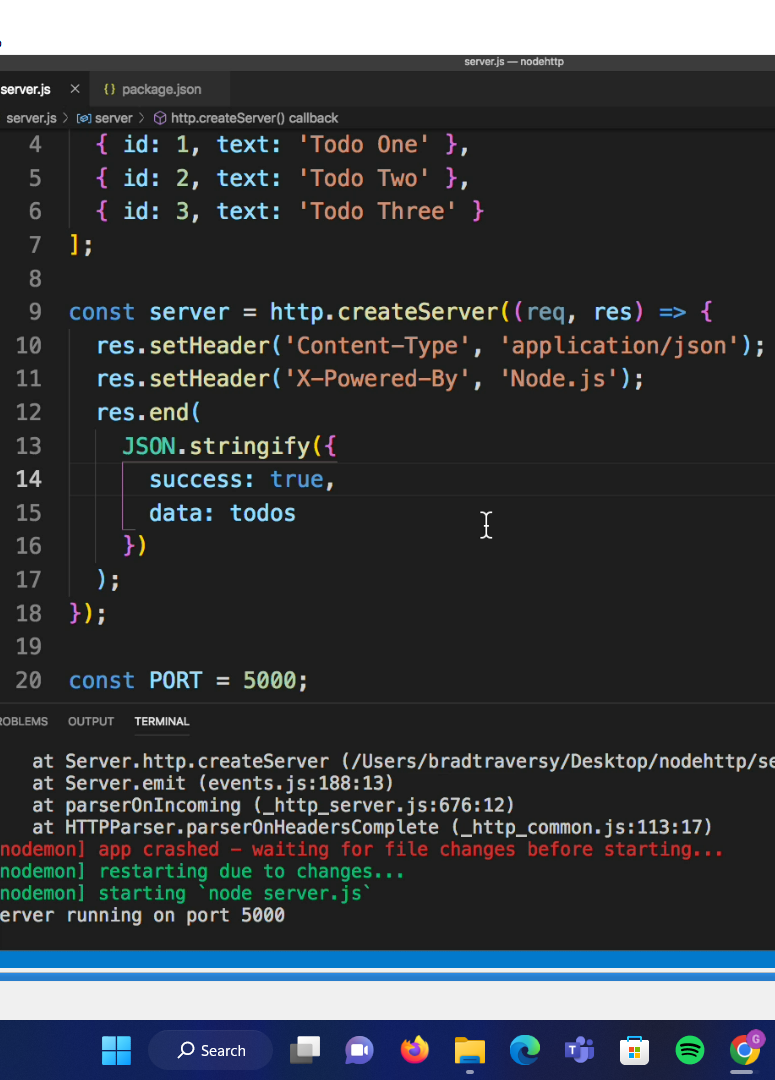
res.end() : it will automatically return 200(ok status) if there is no error. 

In HTTP : we have to do res.end(json.stringyfy (to convert data to strings)) bcoz res.end() doesn’t have inbuild functanility of converting data to string while in express res.send() converts directly the data to string.

To get json data from postman 🡺 use res.json({name: “bread”});

U can also use data from postman 🡺 use res.send({name: “bread”});

Used morgan to directly take calls from middleware : morgan(formats,options)

The format function takes string and contains argument (token,request,response);

Used color npm package to make identigy different things in console

Used bodyParser because if u send request from postman and then the console wont be able to it to the terminal but if you use body parser then it will.

Parse incoming request bodies in a middleware before your handlers, available under the req.body property.

Make the presets: i.e key: Content-type, value:application/json.

For post request: To create something we use Bootcamp.create() method. Use such methods under try and catch block so that request as well as error can be detected.

Remember : if you are adding something which is not in the model and creating it using post method then that particular data will not be send to the database.

GET all bootcamps uses Bootcamp.find({}) to get all bootcamps. Try and catch block is same as the previous post methods.

THERE are too much try and catch block so there is duplicacy so what we do is make a custom middleware(named: async middleware) of try and catch block so that duplicate can be prevented. And also custom error response so that we don’t need to send errors one by one.

i)Creating custom error handler: to do this always pass next(err). This predefined in express docs but will render in html file and therefore we have to make it json.

Next we have a common class for ErrorResponse which contains the parameters of statusCode and message.

We don’t want to pass errorResponse in every catch statement so will make common errorResponse so that we can only pass next(err).

Catching the specific error suppose we wrongly copied the id and doing console log, therefore it will show “CastError”. So now we will pass if statement to pass specific response for respective errors.

Now we have set name (unique :true) therefore we cant use the same name again , so now for this we have to define a if statement for checking the names but error name can be same therefore its better to use Code which we get in our terminal which is unique.

Here we are creating our own custom error handlers while we can use a third party errorhandler i.e expressValidator by which things become easy.

ii) We don’t want to do again and again try and catch for every CRUD API therefore we are using asyncHandler . It contains predefined code available at mongoose documentation. We named it as async.js and then exported it to the every API as a middleware

Creating Predefined Mongoose Middleware: (pre : if applied then runs before execution of a API, post: if applied then runs after the execution of API’s)

If you want to fetch data from backend say from hooks in react.js then you can use slugify . It contains a special option which can be called to the frontend.

Using GeocoderMap: using this api so get the location(latitude and longitude simply by the address). Using the code of mapquest\_geocoder(documention predefined).

Seeder.js is made so that we can directly delete or create all bootcamps,courses,review etc at a one go. JUST TO SAVE TIME

The **process.argv property** is an inbuilt application programming interface of the process module which is used to get the arguments passed to the node.js process when run in the command line

It starts with 2 because the code will be run with

node myprogram.js firstarg secondarg

So

process.argv[0] == "node"

process.argv[1] == "myprogram.js"

process.argv[2] == "firstarg"

getBootcampsbyRadius is an API which we get (/radius/:zipcode/:distance). See the code it is predefined in centerSphere of mongoDB.

Mongoose Advanced Query:

1. Filtering: A **match** is an object which specifies exactly which posts we are trying to match. For example, we could match posts where published is set to true or where published is set to false.

To find in

1. Array use $in . Eg: url?carrers[in]=Business
2. Using operators : url?averageCost[lte]=10000

Filter and Select: ------------🡪>>>>>>>>>>>>



Sort( ):



Pagination: is used to give pages to the respective document. You can select prev and next page to denote no of pages. The advantage of this so that the frontend can fetch results in diffrenent pages. The task will become more easier and you add your link to the frontend.

Pagination is the process of adding the consecutive numbers to identify the sequential number of the pages. It indicates a series of related content exists across multiple pages.

For example, getting enormous data and render as the list only on a single page may increase the latency of your application. So, to break the content and show it across the multiple pages will suffice the requirement of pagination

There are two options which support pagination, limit and skip. They are both are going to be a number. If you want let’s say 10 records, we will need to send the query parameter as /posts?limit=10and the other part skipwill allow you to iterate over the pages: /posts/limit=10&skip=0**.**This will get us the first page.The route /posts/limit=10&skip=10, will skip the first 10 records and give you the second page with 10 records if there are any.

**COURSES IMPLEMENTATION:**

Made the course model and make sure that it has ref:Bootcamp. We have done this because our couses have specific bootcampID and we want to access those courses it should match any of the bootcampID.

In the bootcamp.js(routes) we have added courseRouter and that has parameter of (./bootcampID/courses, getCourses) and then in course.js(routes) we have done (./, getCourses) i.e we have mergerd the both routes in order to work together and last to merge both of them we have used ({mergeParams: true}) in course.js(routes) because we want to merge both of them.

Very Imp:

Populate :

1. When send request to get all courses we get the details of only the courses details + bootcampID. But we also want the information of bootcamp. Heres populate comes to the picture.
2. If we want ALL information of courses along with bootcamp then simply concat populate(‘bootcamp).
3. If we want only specific info of a bootcamp(like only the name and the description) along with courses, then

 query  = Course.find().populate({

            path:'bootcamp',

            select:'name description'

        })

Virtuals(reversePopulate):

1. Now if we want the reverse thing i.e get all courses in our related to that bootcampID. Ofcourse we will make use of bootcamp routes.
2. Goto bootcamp and select route getallbootcamps.find().populate(‘courses’)
3. if u want certain fields only in courses then add path+select as done above.

If we want to delete the bootcamp then all the courses should also get deleted.

1. First of all, if we want to delete the bootcamp so that all courses get deleted then there is a remove(), but make sure in the DELETE method of your bootcamp API u use findById and not findByIdAndDelete. Further call bootcamp.remove().

Since we have to remove there we have use findById() and then bootcamp.remove().