We are using a cloud version of MongoDb that is the atlas and that is because we want to host the final project on the Heroku which is a platform as a service and hence doesnot really allow putting the local db on it.

We used the AWS cloud service for Mongo cloud.

Database name: DevConnector

In the database , use ipwhitelist as its global so that is like just giving access to your own ip address.

Now we will be also using mongoose, this is just used as it gives us some methods to play with our database like .find()

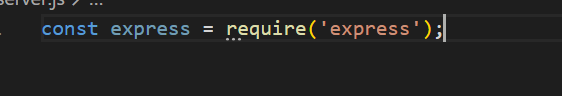
Backend: using ExpressjS

Npm init, in terminal

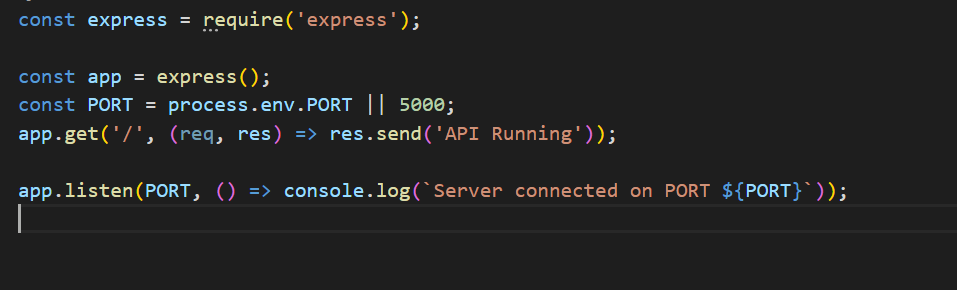
Using bcrypt for password hashing, config for global variables, gravatar for profile avatars, jsonwebtoken, mongoose, request

We use dev dependencies like nodemon(so that we do not have to refresh the server everytime we make a change), ‘concurrently’ which will allow us to use the react server and node server at the same time.

Create a new file server.js, once u have that and now you want to load the dependencis in this file that u just installed above. Here is a sample how u do that :

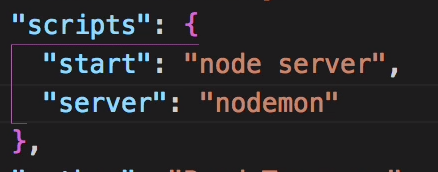


Now we will put our app variable as express and then make this app listen on some port.`



Well this is hiw the general nodejs backend runs, first u get the app as express framword, set up the port for listening, in here we put PORT as process.env.PORT i.e bcz this will take up the port number once we will put the app on Heroku environment. Otherwise for our local we also put 5000. Then we put the get function this is exactly same as that we have in java servlet. Set up the callback function with req and res as parameters to handle the reuests and responses. res.send() is same as out.println() in java servlet.

Now u can run this using npm start, (but first go to package.json and addd a script for keyword start.. i.e edit the test vla block and put start [test di jga paayi ] and then in front of start : see example=>)



Later on we will add the client script as well to run the react amd them the dev script to run these both concurrently.

After this for now run In terminal: npm run server

We are using postman for all are requests.

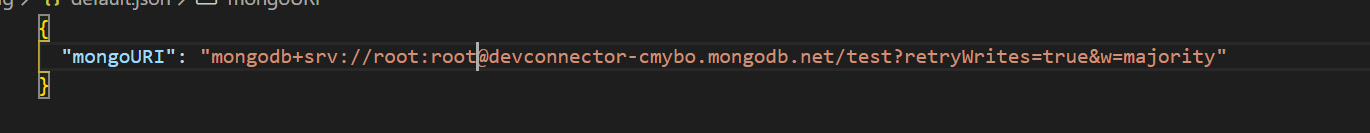
Go to postman application and put up the request address:



Hit send.

**Connecting to the MongoDb:**

Create a folder config in project folder. Config will be used for those things which are common throughout the project like the mongodb url. Create a file in config names *default.json.*



Uri ohi hai jehdi asi copy kiti si mongodb atlas tch connect te jaa k.

Once done create another file in config folder named Db.js.

In this create a method connectDB , this is the method that we will call from the server side while connecting to the db.. same a VMMDBLOADER :P



We get the mongoose package and we also fetch the config package as we need to use the default.json

Using config.get() you can fetch any of the attribute of the json object.

Then we create the connectDB method.

Now notice that mongoose.connect(db) will be used to connect to the db and it wil return a promise. So we will use **async and await** through out the projet cz that is the new convention being used in industry.

The word “async” before a function means one simple thing: a function always returns a promise. Other values are wrapped in a resolved promise automatically.

And the keyword await is used to let the javascript know that we are waiting until the promise settles and gives us the response.

In case there is a problem while connecting to the db we just abort the process.

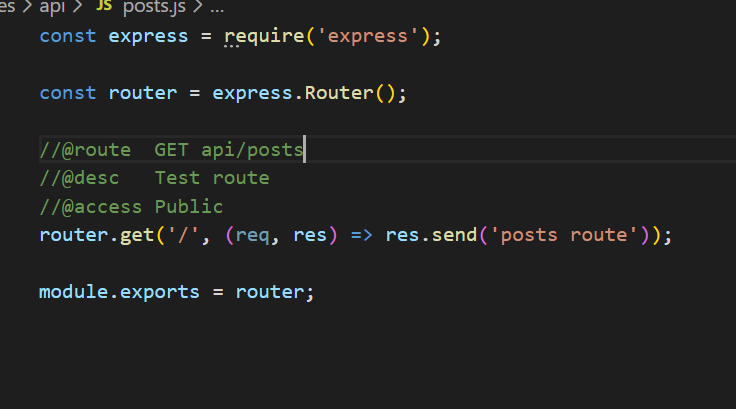
At last we export this function as we need to use it in other files.

**Router:**

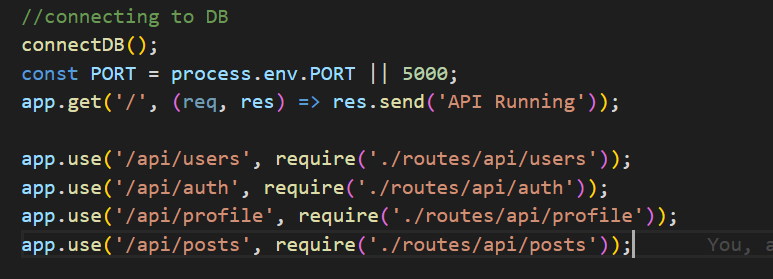
Create a folder routes and these routes will return json for the api’s so lets create a folder api.

And as we know that are project will have main 4 things: profile, users, auth and posts.

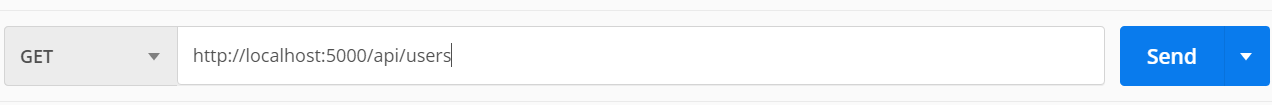
So we create 4 js files and set the router as:



Now to take care that the request comes (routes here ) when intended go to the server.js back and put:



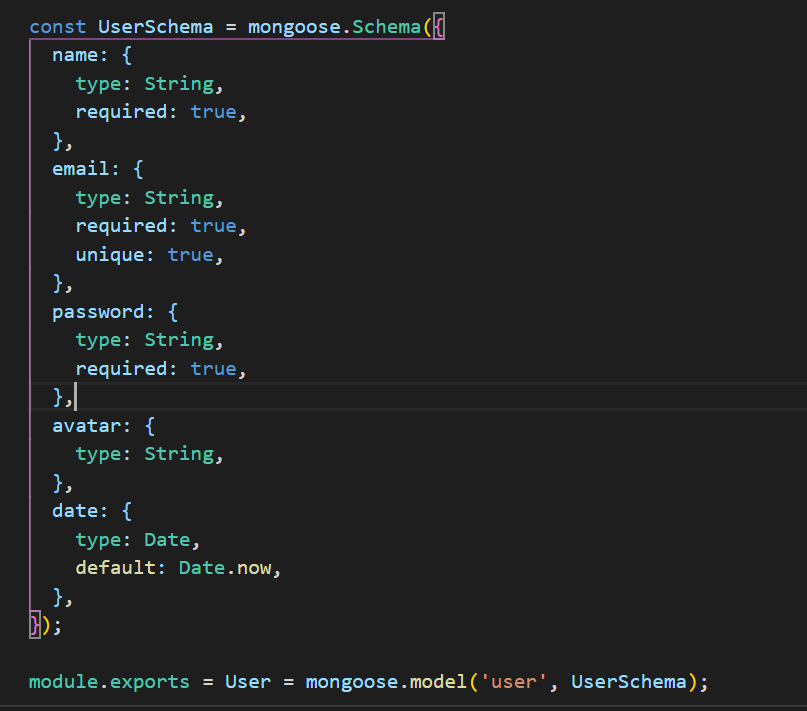
This basically tell where to go when a request for a certain page comes . Now to make sure things are working fine, go to the post man and tr making a request->



Now creating the **USER:**

In here we will make the user model and mongoose model which will hold all the data. So here in order to connect to db, we need a model there fore we create a model folder and first create a User.js (capital U).

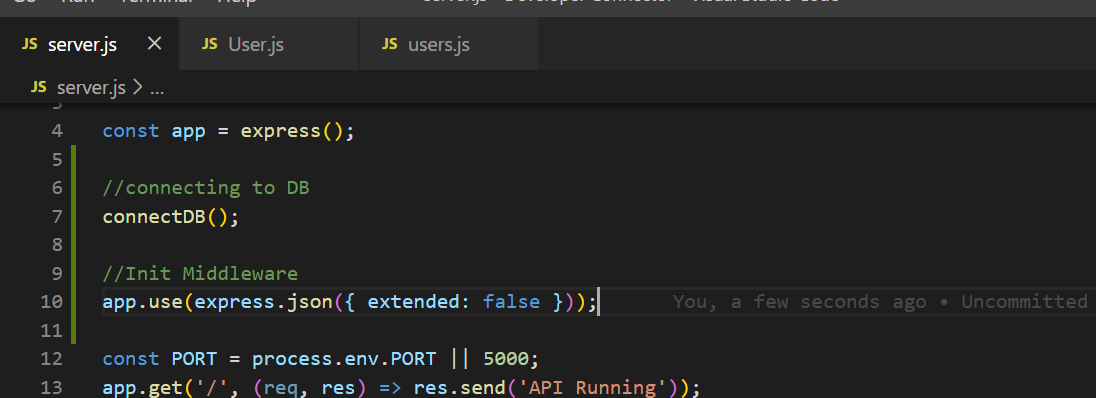
In that we first make the schema: UserSchema and pass on the fields in it.



Mongoose.schema takes up the object of fields with their specifications.

And once done we do module.exports = User = mongoose.model(‘user’, UserSchema);

user here as the first param in mongoose.model function is the name of the model



Middleware setup so that the data can be fetched in the req parameter of users.js

Go to post man for a test request.

Now once we start to get some request we need to validate the name, email if its correct or not so we use express-validator.

Once validations are set up the order of workflow is going to be :

//check if user already exists

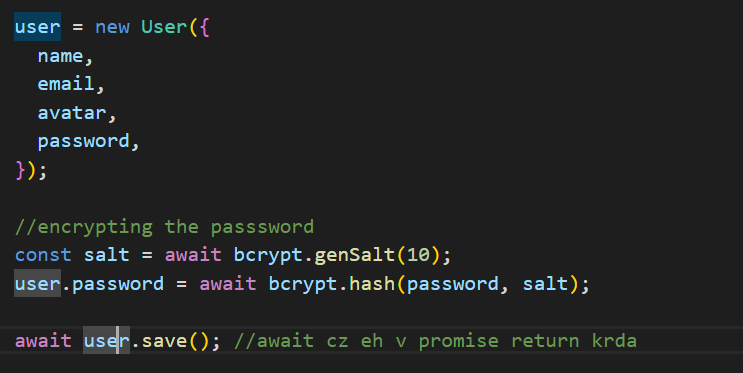
//get gravatar using email

//encrypt the password

Bcrypt send a promise so we use await while calling it and use the function genSalt(10) to get the salt for the password.

Use the function bcrypt.hash(password,salt) to encrypt the password.

Once the password is correct we create the entry in the mongodb ->



Now if u remember wadda U vla ikk User.js bnaaya si jide ch model prepare kita si ,, well we make one object user of that class, add the name, email, password, avatar in that and save it… on save it makes the entry in the mongoose and is a promise function which will return the attributes . In mongoose the \_id of mongoDB is abstracted as id , hnce in case u want to use that id you straightway do user.id

// return jsonwebtoken

Jsonwebtoken oh cheej aa k eh saare tuhaade object nu lae k ikk token bna denda so that baach jdo user login krega assi token toh dekh skaange kon aaya and kithe osdi profile load krni bla bla



This is like keeping things in session, to know who is logged in and if active for valid time in session.

So we use the jwt.sign() function which takes three parameters and return a call back function.

1st param: payload, it is the data of the person.

2nd: jwtsecret, set this anything. (kept it in config default.json)

3rd: time for which you want the person to be active

Then it returns the call back function which gives wither the error or the token itself. Well if error then throw that error otherwise make json object of taken and send it bak

Now we create the middleware to authenticate the token that we just got back

Create new folder middleware and Create a file auth.js in this folder.

So this is a middleware function now we keep 3 parameters, req,res and next

Understand this thing that tokens are sent in the protectd https requests while traversing the pages (jive u see fb te hyperlink kaafi strong encrypted hunda token is the same thing)

So in order to get that token here , we have req… use the function req.header(‘key’);

Key is like ./profile?key=”token tera ethe saara”

Once we get the valid token in here, we use the jwt.verify(token, jwtsecret) function this gives us the decoded f=form of user and we put that in req.user so that hunn assi esnu kite v use krskiye.

Next() har middle ware ch use hunda hai once ur done with ur hisaab kitab just put next() at last

**Now the mainly 2nd step starts : Authorization**

Go to your auth.js file that we created at the beginning and import this middleware auth.js there.

Once u import this ‘auth’ then go to the get function jehda pehla hi othe haiga and put this ‘auth’(middle ware vala ) at the second parameter position.

This basically means jdo request es mage tea aye , then first middle ware te jaaye and j sab theek hai then aa page aggo challe

Now next is the post requet in same class:

This is for login purpose and in here we check the email and password of the user.

To check the password we use bcrypt da compare password that takes 2 parameters: 1st: plain password

2nd param: encrypted password.

First user is inserted using user.save()

Baach profile update krn vele: Profile.findOneAndUpdate({user: req.user.id}, ($set: });

3rd again creating profile time v await profile.save();

For fetching all the profiles we have used the find() function . Remember findOne is used to get a single record and find() for multiple records.

**Deleting the user and profile.:**

For this we use the delete request. i.e **router.delete();** (its like the same as get and post request)

We use the method Profile.findOneAndRemove(user id), user id is fetched as we have jsonwebtoken in the request and hence we use req.user.id to fetch that id and delete it.

Add Experience and Education:

Experience: we are using the put request in here(can use post also no issues.)

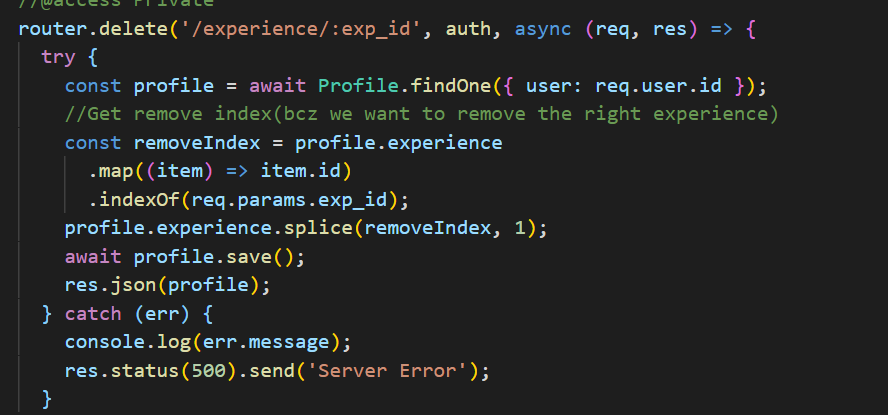
frst we find the profile of the user the same way using the id: profile = Profile.findOne(user: req.user,id);

profile.experience.unshift(newExp)

newExp is the object that we created for all the entries. Unshift method is same as push [experience j tenu yaad hove te ikk array bnaaya c mongodb ch]. OS unshift is like push but only difference is push last te add krda… unshift means most recent the first.

Then profile.save();

**Deleting the Experience:**



**Get Github Repo for profile:**

So for this we made our app registered with github.com. We did the Oauth app register in Github.com dev settings.

Once you do this you get the Client ID and Client Secret

Take this is and put it in your config-> default.json

So we make an api request to github.com asking for the profiles and this is how we send that->



We make an object options, we keep the fields as :

Uri: whole link of github including parameters like the username, the repos per page, sorting order as ascending and then we pass the ClientID and the clientSecret as well that we save in the default.json class.

Second parameter is : method: ‘GET’

Third: headers: {‘user-agen’: ‘node.js’} idk why we passed this though

Once complete the object options , now we finally send a request.

Request() this is the package that we installed in the first lec (at that tym u said react and server ikathe chlaan lyi hai well pta nai pre the useho rhi).

Request takes 3 parameters, 1st: options object, 2nd: response(should be 200 tells request successful same as in javascript request krde si aapa ), 3rd: body. Body is nothing but the return whole value from git

This body is in the form of a string hence needs to be parsed into JSON bfr sending out.