Resolvers

we are verifying JWT on all requests which are requesting GrapphQL Queries and Mutations, if the verification is success then the user id in the payload will be attached to the request. which is usefull in queries and mutations

Index,js is the rootResolver with connects all these resolvers in different files to Rootresolver.

Resolvers in userresolver.js

getUser: this is a Query similar to get request, It gets the user Information with users id= request userid, It will check the userid in the userModel in MongoDB and return the object with information when it finds. if not found gives error.

updateUser(userInput): this is a mutation similar to post request, in this, the information that has to be updated will be in the userInputwe will take all the information from the userInput and updates the user information by finding the user with the request user id in the userModel in MongoDB. and using findOneandUpdate mongodb query we will update this users information with information in the body of request.

profilePic(avatarUrl): This is a Mutation,the avatar url is passed as a avatarUrl Argument. at first we will find the user with userid in request in UserModel and updates the avatar field in the model with this url, this also by using findOneandUpdate query in MongoDb

getProfilePic:This is a Query, it returns the profile picture. First I am find the user with request user id in user model. and returns the avatar in the avatar field of this instance.

Resolvers in subscriberesolver.js

getMynewspapers: This is a Query, it returns all the subscribed newpapers of the logged in user, I am searching all the subscribed newspapers which have user id as the request userid. So there will be array of such subscribed newpapers. this can be done by using find MongoDB Query on MyNewspaperModel with user = request userid. and populating each subscribed newspaper with its newspaper title and avatar, avatar's avatar, like we will store the id of newspaper and avatar, so we will populate them as they are refering to their respective Models. this can be done by .populate() thing. populate is for easy use in frontend instead of doing again request for these ids.

subscribeNews(newsInput): This is a Mutation, It takes the infomation in newsInput, and and make a MyNewspaper model instance with the user field as request userid, and with information in newsInput. and saves it in database. Now we have to increase the noofsubscriptions for the avatar of avatarid in newsInput, as he is subcribing this avatar the subscriptions will increase by 1, so by using findOneandUpdate query for AvatarModel with this avatar id we wil increase the subscriptions by 1, similarly for newspaper with newspaper id in newsInput we will use findOneandUpdate Query for NewspaperModel with this newspaperid we will increase the subscriptions by 1, these subscriptions will be used in sorting showing popular things on dashboard. returns the saved information.

updateMyNewspaper(newspaperid,newsInput): This is a Mutation, it takes the Information in newsInput, Now we have to update the information of subscription of a Newspaper with Newspaperid =newspaperid.

For this we will find the subscribed information about this newspaper and updating the information which is in newsInput. we will use findOneAndUpdate with newspaperid=newspaperid, and the updates the information. if the avatar is changed then we have to increase the subscriptions by 1 for the changed avatar and will decrease the subscriptions by 1 of the before avatar. if it is same then we will not do any change in subscriptions. returns the updates one.

getMyNewspaper(newspaperid): This is a Query, I am searching the information about subscription of newspaper with id = newspaperid. I am using findOne query on myNewspaerModel with newspaperid = newspaperid,it returns the information of the subscribed newspaper. Here we will populate the newspaper id with its title and avatar, the avatar id with avatar, to make easy in frontend.

deleteMyNewspaper(newspaperid): This is a Mutation, I am searching for the newspaper with this newspaperid and deleting it. this is done by findOneAndDelete query on MyNewspaperModel with newspaperid=newspaperid,After that we are decreasing the subscriptions by 1 for the avatar and newspaper in this deleted information of subscription of newspaper.

setAvatarforNewspapers(avatarid,newspaperids): This is a Mutation, newspaperids are array of ids for which we have to change the avatar id with the passed argument avatarid in subscription information of particular newspaper. for this we are looping through all newspaperids, and changing the avatarid to this avatarid by using findAndUpdate query for each newspaperid. in each loop we are decreasing the subscriptions for the before avatarid by 1. After updating for all ids, we will increase the subscriptions for the avatarid given in argument by no of ids given in argument.

delUploadedAvatar(avatarid): This is a Mutation, this deletes the uploaded avatar, with id == avatarid. for this we are using findOneAndDelete on Avatarmodel with avatarid==avatarid.

Resolvers in awsresolver.js

signS3(filename,filetype): This is a Mutation, In this we will get the signed Url and url for this file which will be uploaded. first creating a instance of aws S3 with AccesskeyId, secretAccessKey, signature version and region. we will a object s3params with bucket name,key as a filename, ContentType as filetype etc., we will make a signed url as `s3.getSignedUrl('putObject', s3Params)`; in this way we will get the signedUrl, this signedUrl is used when Uploading this file we have to send this signature along with the file. we will create the url for the file to be uploaded. this returns signed url and url

delProfilePic(filename,filetype): This is a Mutation, In this we will delete the file with the filename in s3 bucket. here are we will initialise aws s3 and s3params, now in s3 params only bucket name and key as filename. here for deleting we will do `await s3.deleteObject(s3Params).promise()` this will delete the file in s3 with the passed filename.

Resolvers in allresolver.js

allNewspapers: This is a Query, In this resolver, firstly, I am finding all the newspapers in the database and subscribed newspapers using the mongoose find function. Then I am finding all the subscribed newspapers of this particular user. After this, I am mapping through all the newspapers and checking if they are subscribed or not. If yes, I will put the newspaper itself in the array; otherwise, I will put false. Now I am filtering the array we got and returning if not empty; otherwise, the 404 status code is returned. If there is any error, then also 404 is returned.

allAvatars: This is a Query. In this resolver, firstly, I am getting all the avatars in the database. Then I am checking through all the avatars. If this has a user field null (preadded avatar), select it, or if this user uploads this particular avatar, then also select it. Then I am returning them.

popularNewspapers: This is a Query, In this resolver, I am doing the same thing as I am doing in the all newspapers resolver. Firstly, I am finding all the newspapers present in the database while sorting them in descending order based on the number of subscriptions. Then I am selecting all the newspapers subscribed by this particular use. Then I am filtering through all the newspapers, selecting only those which are subscribed by this particular user using map and filter functions. Then I am slicing the array of newspapers. I got to select the top eight entries only and returning them if the array length is greater than 0.

popularAvatars: This is a Query, In this resolver, I am selecting the top 8 avatars by sorting them according to the number of subscriptions, and then I am returning them.

getNewspaper(newspaperid): This is a Query,In this resolver, I am finding the newspaper with the given newspaper ID. If it exists, return it; otherwise, an error is returned.

getAvatar(avatarid): This is a Query, In this resolver, I am finding the avatar with the given avatar ID. If it exists, return it; otherwise, an error is returned.

addNewspaper(newspaperInput): This is a Mutation, In this resolver, firstly, I am creating an instance of a newspaper using NewspaperModel, adding all the details like the avatar (URL), title, and description about that newspaper. Then, I am saving that newspaper instance into the database so that it can be used later. Then I am returning that saved newspaper. This is only for populating the database purpose. user from frontend cannot call this mutation

addAvatar(avatarInput): This is a Mutation, In this resolver, I am first checking the **if user** field in the avatarInput, and if it is not equal to "a," I will create the new avatar instance using the AvatarModel will have a user field equal to the user ID of the currently logged in user; otherwise, user filled null, so that this avatar is a preadded avatar. Then I am saving this instance of AvatarModel to the database.This is only for populating the database purpose. user from frontend cannot call this mutation

In all these resolvers, I am using try/ catch so that if there is any error while doing all these things, it will throw that error, and a 404 response is sent to the user or whosoever made the request.