Folder structure

1.frontend :

1.create vite project (npm create vite@latest ./ )

2. select react and javascript

3.We can start the server by npm run dev command

2.backend

->create .env file

->create .gitignore file copy contents of .gitignore from frontend .gitignore file and paste in this and along with that add .env in that.

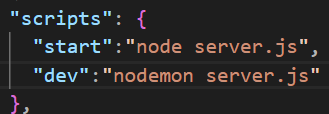
1.initialize npm (npm init)

2. we can now install packages for example express, nodemon ,etc.

->In backend we will use import to import modules,packages. So go to package.json file and add

“type”:”module” below name field.

->In scripts field remove test and add two more fields



->whenever will run command npm run start then this “node server.js” command will run or when we run command npm run dev then “nodemon server.js” this command will run.

Now install some packages.

bcryptjs, cookie-parser, dotenv , jsonwebtoken , mongoose, socket.io

\*\* Now push project repo to github

After that create a new project of chatapp on mongodb atlas and copy MONGO\_URI and paste in .env file

->Now connect our backend server with mongodb database



->Now add models folder in backend

1.add user model

MONGOOSE VALIDATORS

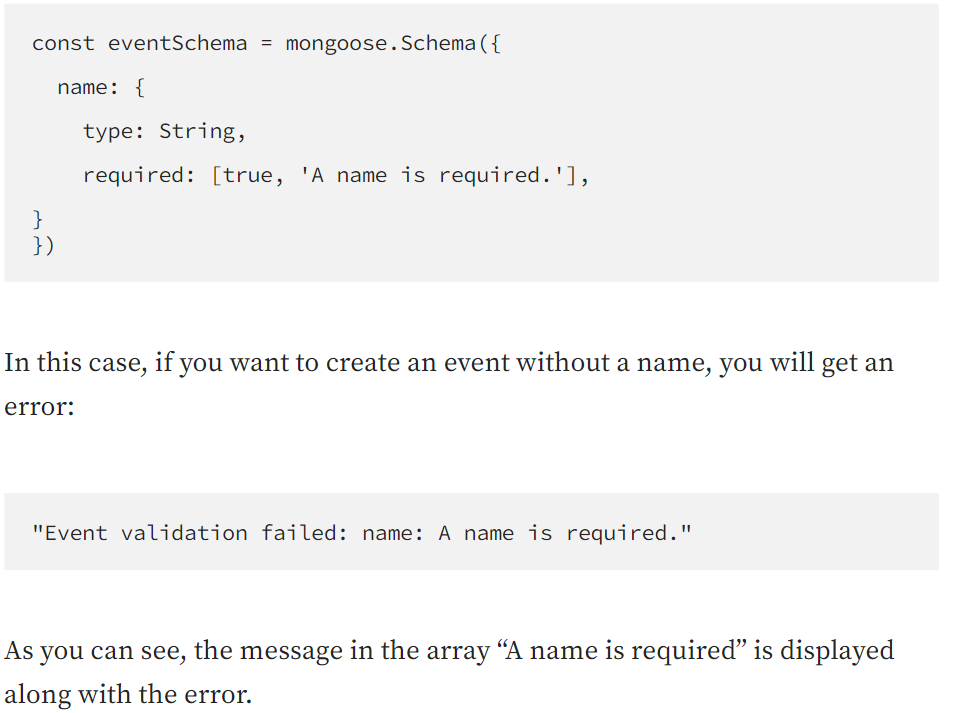
Two types of validators :1.buit-in 2.custom

->***All SchemaTypes****have the built-in required validator.*

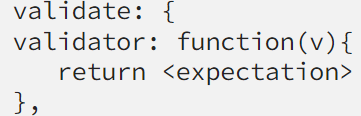
***Numbers****have min and max validators.*

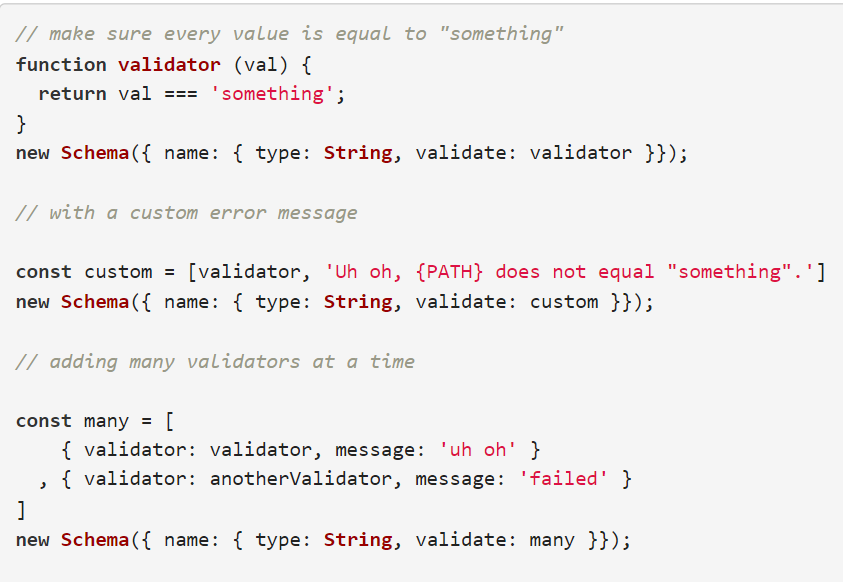
***Strings****have enum, match, minlength, and maxlength validators.*

The validator usually looks like this: validator: [expectation, message] . The expectation will be something you expect, and the message will be something you return along with the error.

**

Custom validators





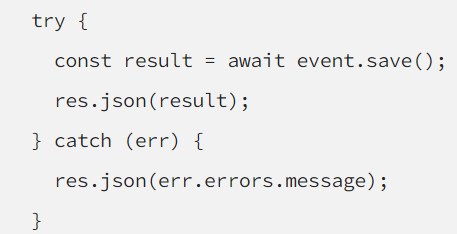
ERROR Handling of mongoose validators



For this error will be



=>so to handle this

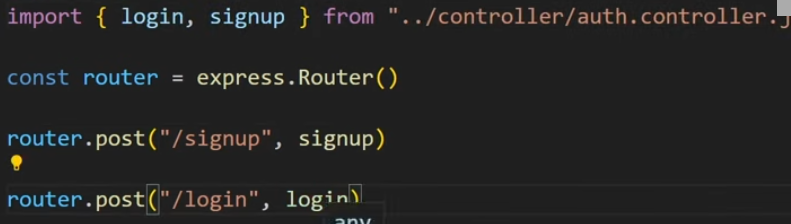


=>now create a routes ,controller folder in backend.

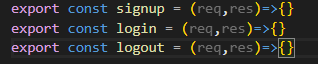
=>routes folder contains all of the routes of backend

=>Implementation of these routes function is done in controller for particular routes

=>for example for auth route(auth.route.js) it contains routes at signup,login,logout and the functions which this will execute on these route they are defined in controller , auth.controller.js file which is inside controller folder



Like this

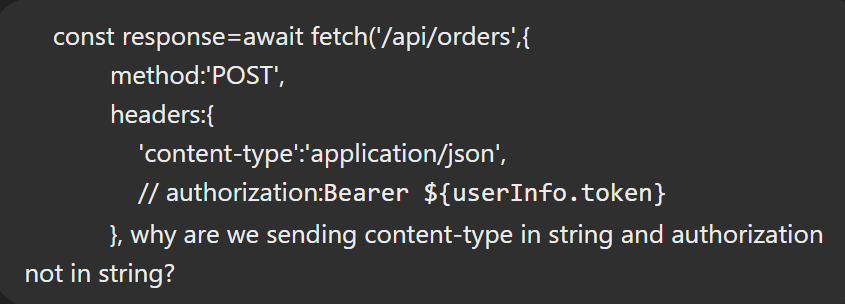


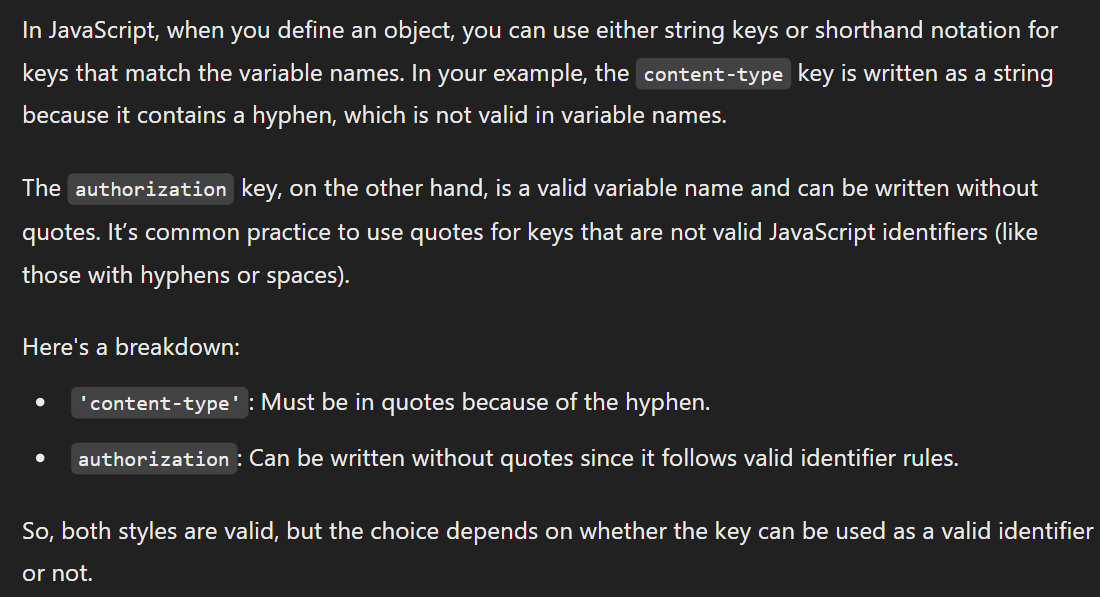
Inside auth.controller.js file

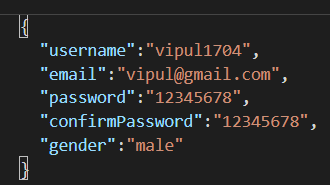
We can give avatar to user once they signed in. So for that we have to use random avatar api

=>We have added a avatar field in user’s xschema

1.go to Avatar Placeholder







If we are sending this data in post request then it will be okay but if suppose it is like

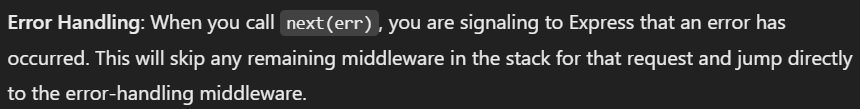
…. ,

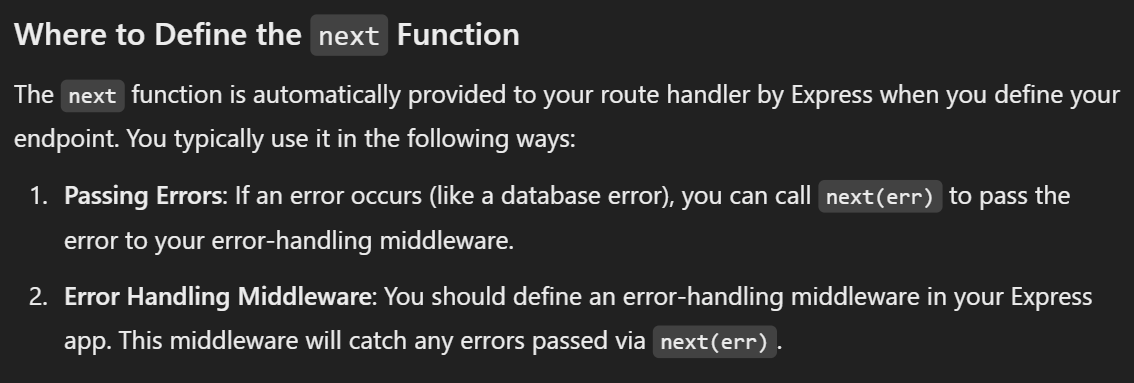
“gender”: “male”**,**

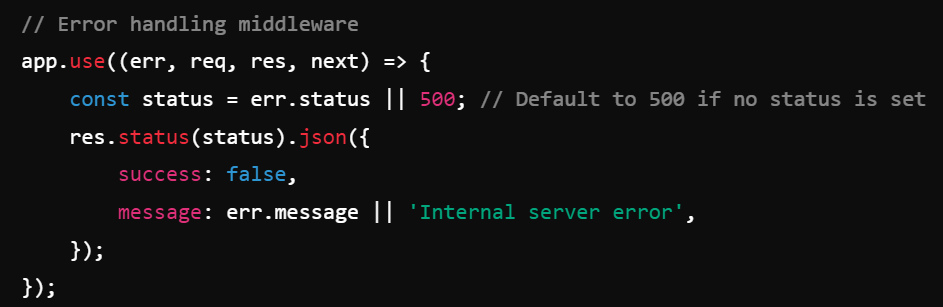
}

**Now it will give error. “ , “ after last property is invalid JSON**

**=>Don’t include , after last key: value pair**

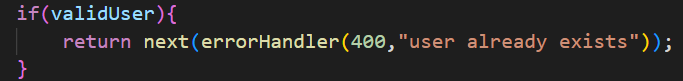
****

****

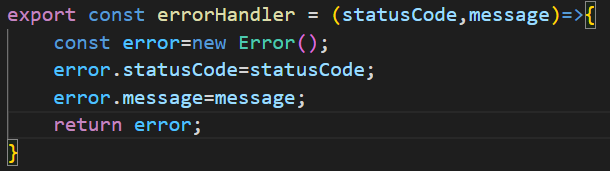
****

**=>This is called as error handling middleware .**

**=>in server.js file you should write error handler and that also at last.**

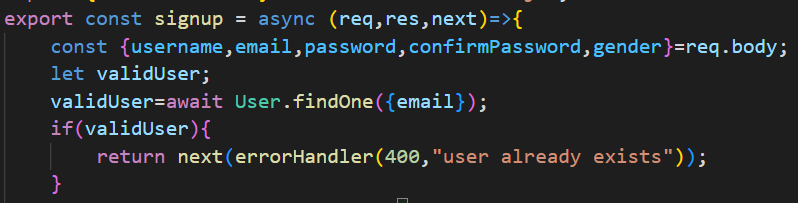
****

**Making error Handler function**

****

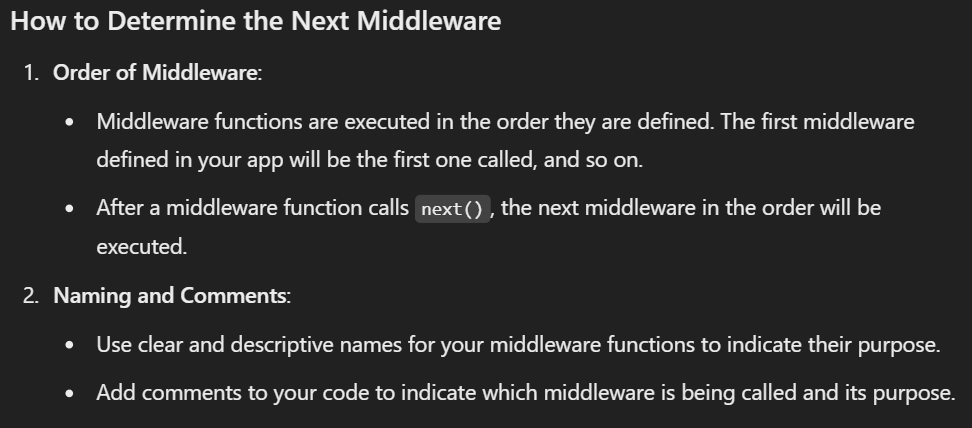
**=>now errorHandler is returning an error**

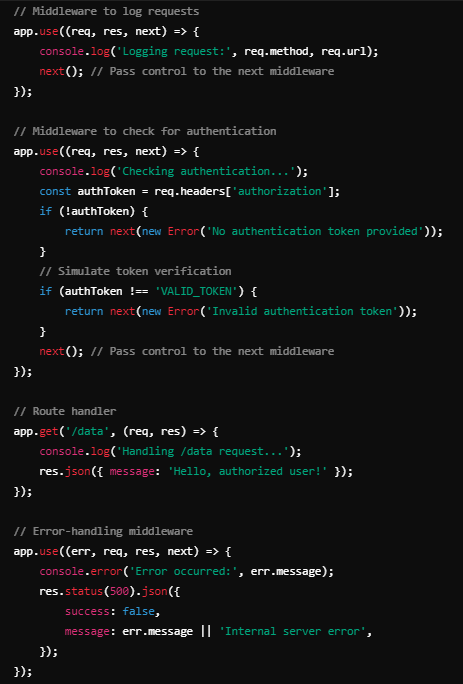
**=>it means next(err) err is the error returned by errorHandler function now we know whenever error comes in next() function then it directly passes the control to error handler in server.js (main server file).**

****

**=>to use next ,we have to take it in parameter.**

**MIDDLEWARE**

****

****

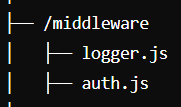
**=>now here first middleware function which will get’s called is first one which is defined at starting (log request ) bz it is coming first ,so it is getting first order.**

**=>when this middleware function calls next() function inside it , it means to call the next middleware function in stack. And next one is middleware to check authentication , so control now passes to this function.**

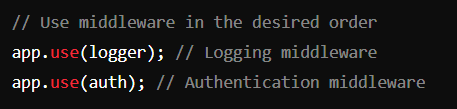
**=>in between if in any middleware error comes then we will use next(err) function to pass the control first to error handler middleware.**

**=>So this is how a series of middlware functions get’s executed in request –response cycle using next() function.**

**=>We can define middleware functions like this also but order of execution will be same (in sequence in which they have been defined).**

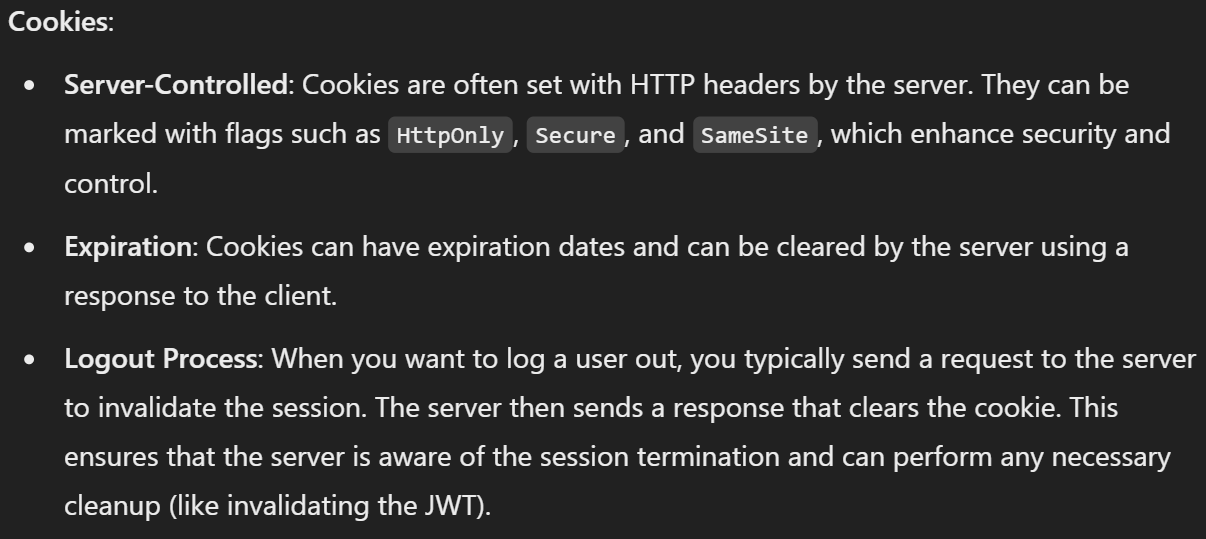
****

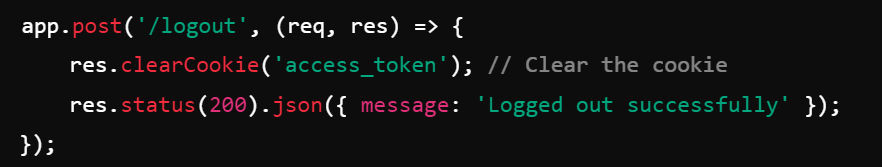
**=>if we create another folder named middleware. And inside that defined those middlware functions are defined then**

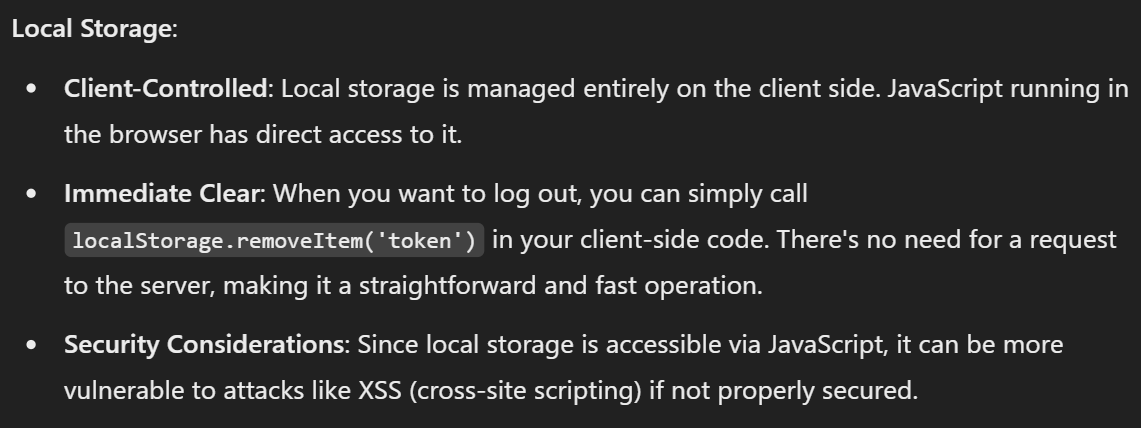
****

**We can use them in server.js file like this . Order matters**

**Cookie vs Localstorage**

****

****

****

**If**

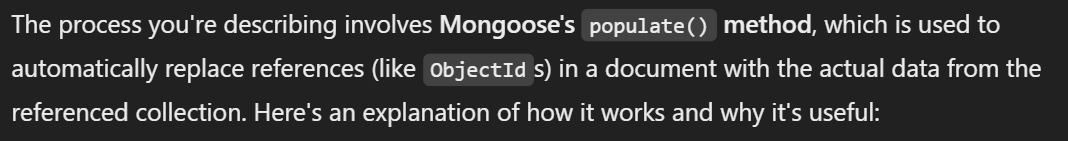
**await conversation.save();**

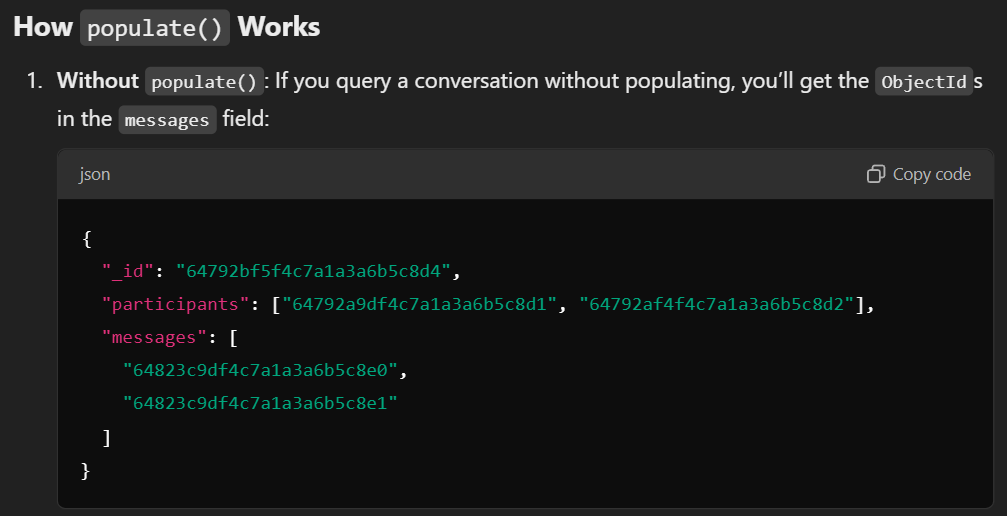
**await message.save();**

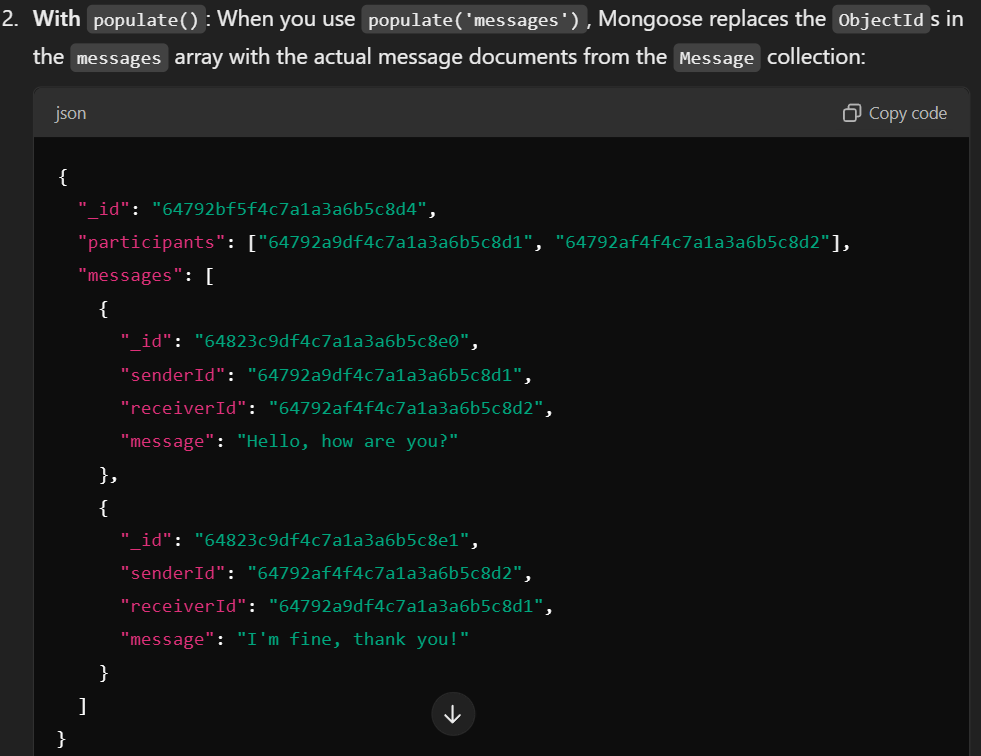
**=>If we are writing this two lines one after other then it is not a good practice. It will affect performance. Instead**

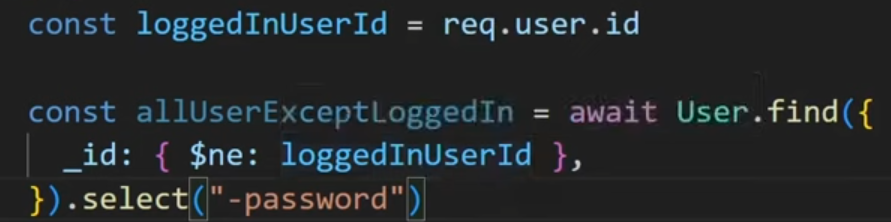
**=>await Promise.all([conversation().save,newMessage.save()]);**

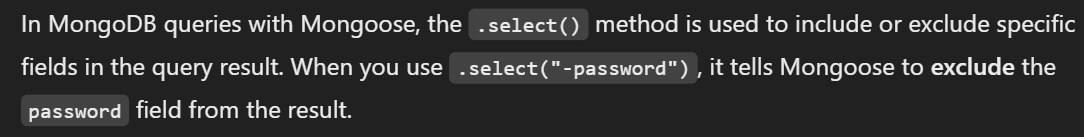
**populate() function of mongodb**

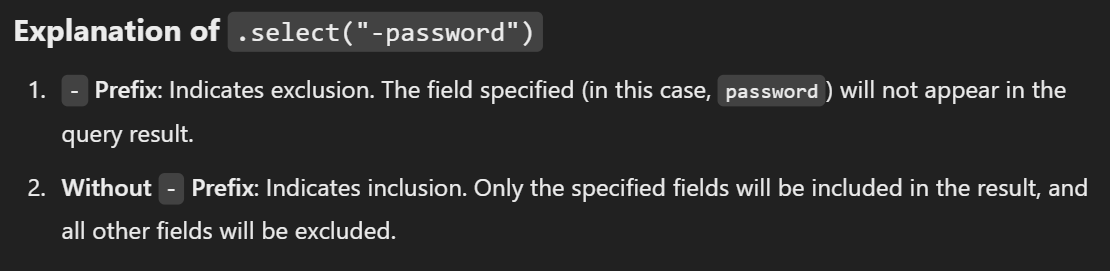
****

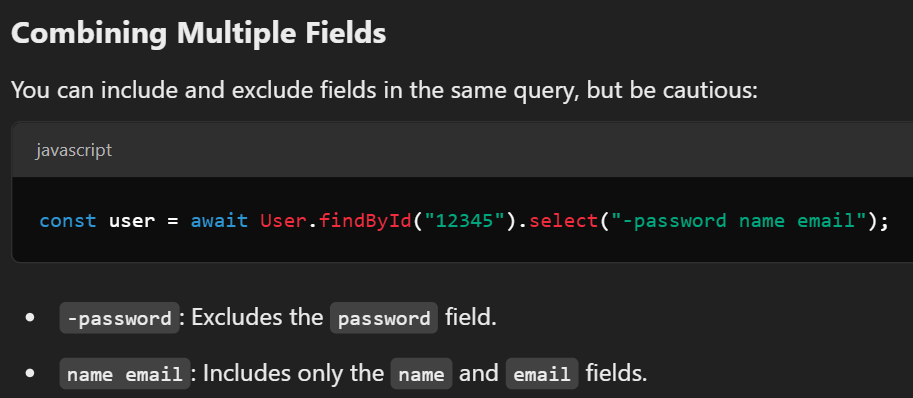
****

****

****

****

****

****