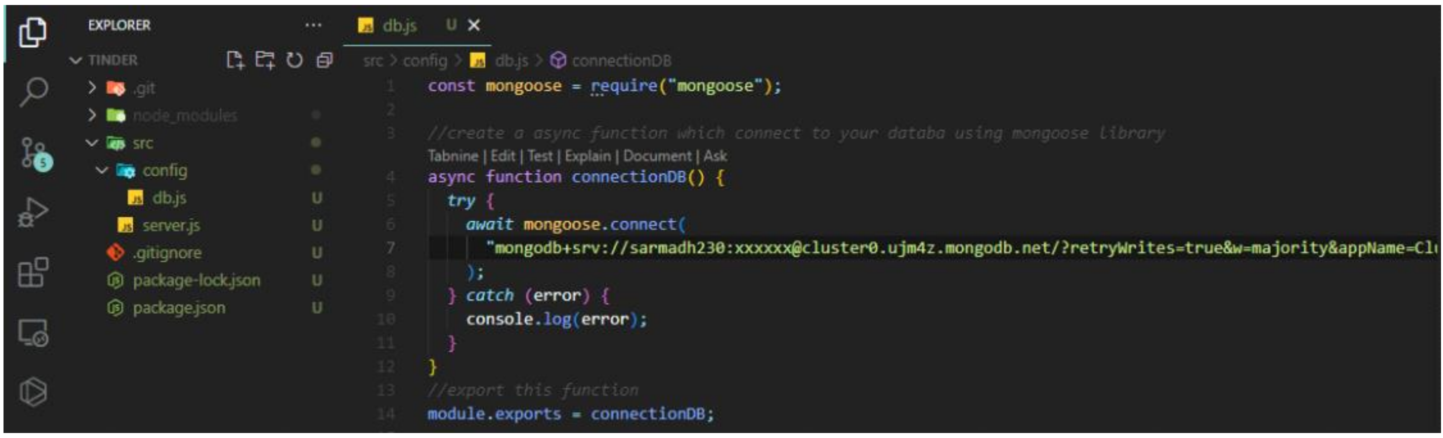
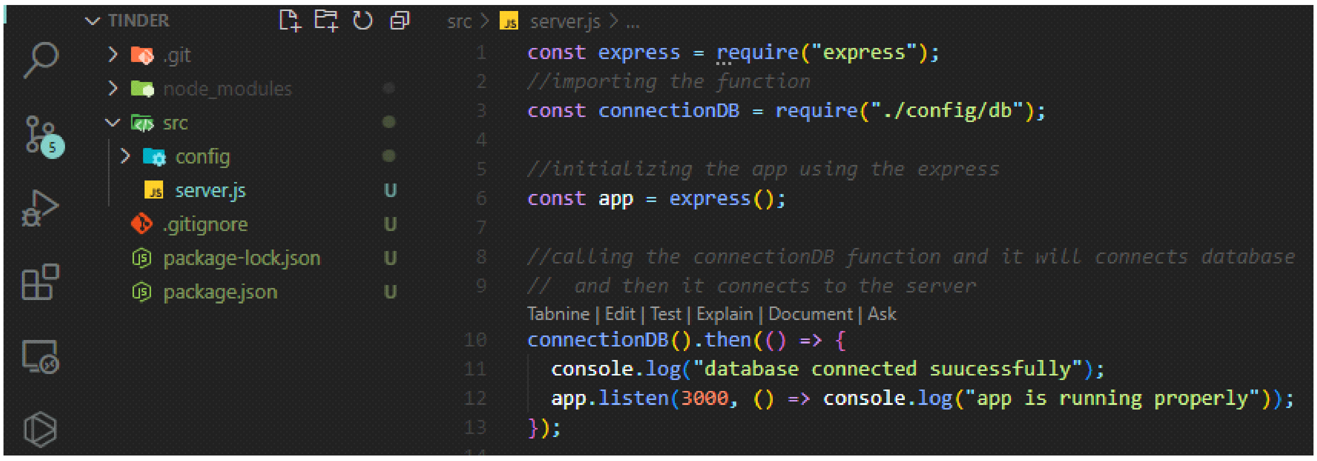
App setup

Monday, 16 December 2024 11:30 am

How to connect database to your application?

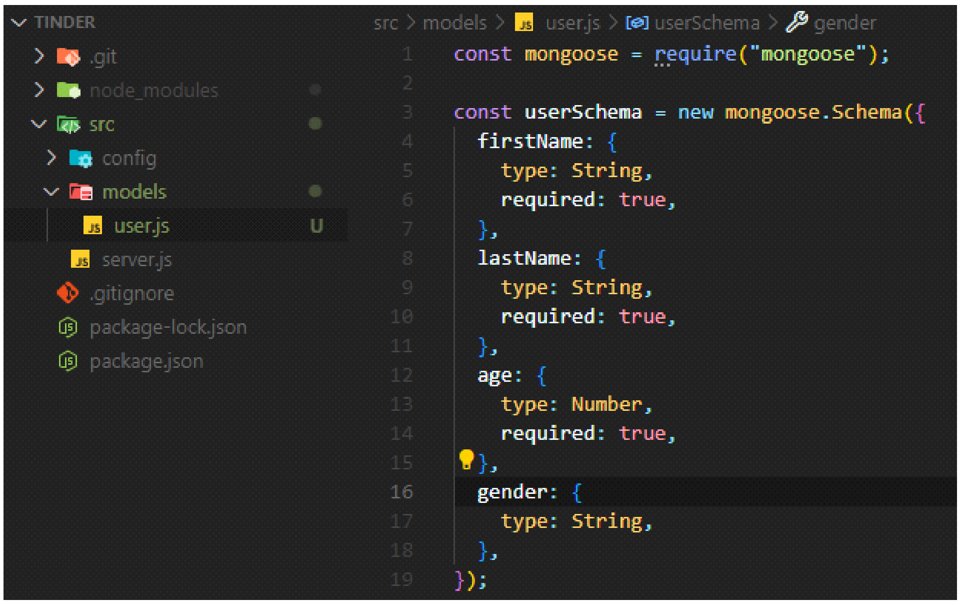


```
const mongoose = require("mongoose");

//create a async function which connect to your databa using mongoose library
async function connectionDB() {
  try {
    await mongoose.connect(
      "mongodb+srv://sarmadh230:xxxxxx@cluster0.ujm4z.mongodb.net/?retryWrites=true&w=majority&appName=Cl
    );
  } catch (error) {
    console.log(error);
  }
}
//export this function
module.exports = connectionDB;
```



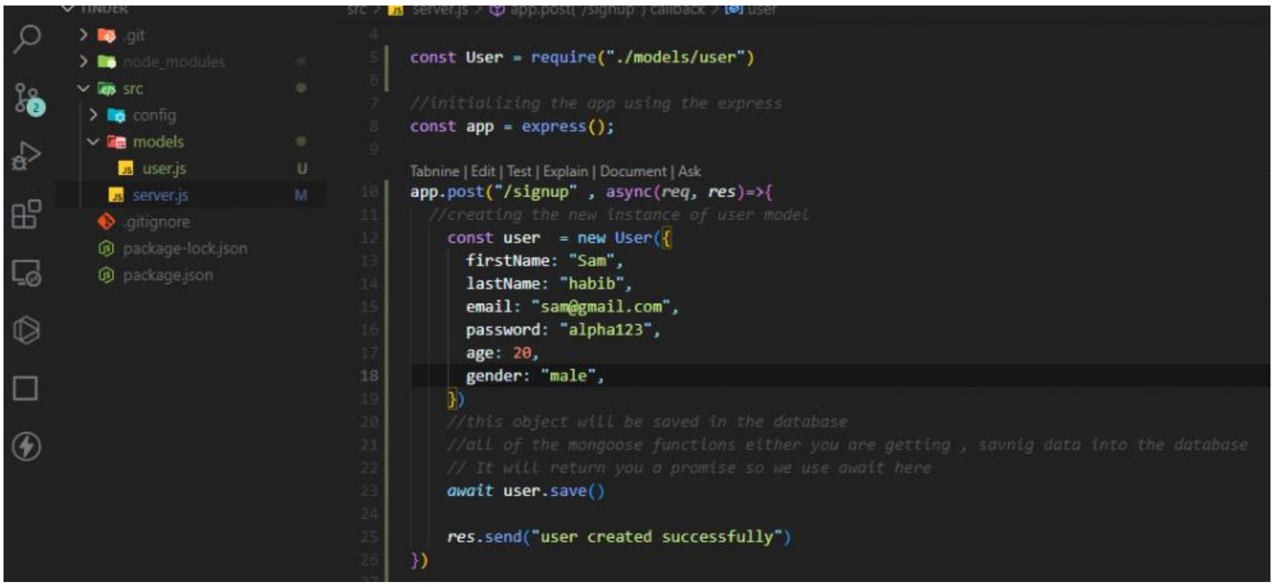
```
const express = require("express");
//importing the function
const connectionDB = require("./config/db");

//initializing the app using the express
const app = express();

//calling the connectionDB function and it will connects database
//  and then it connects to the server
connectionDB().then(() => {
  console.log("database connected suucessfully");
  app.listen(3000, () => console.log("app is running properly"));
});
```

How to create an schema object?

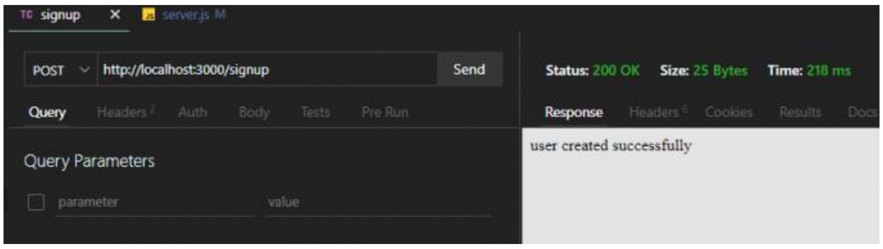


```
const mongoose = require("mongoose");

const userSchema = new mongoose.Schema({
  firstName: {
    type: String,
    required: true,
  },
  lastName: {
    type: String,
    required: true,
  },
  age: {
    type: Number,
    required: true,
  },
  gender: {
    type: String,
  },
});
```

How to make an api call to the database?

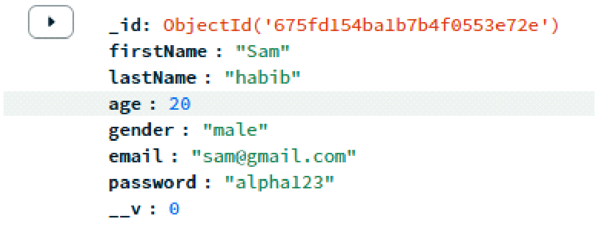


```
const User = require("./models/user")

//initializing the app using the express
const app = express();

Tabnine | Edit | Test | Explain | Document | Ask
app.post("/signup" , async(req, res)=>{
  //creating the new instance of user model
    const user  = new User({
      firstName: "Sam",
      lastName: "habib",
      email: "sam@gmail.com",
      password: "alpha123",
      age: 20,
      gender: "male",
    })
    //this object will be saved in the database
    //all of the mongoose functions either you are getting , savnig data into the database
    // It will return you a promise so we use await here
    await user.save()

    res.send("user created successfully")
})
```



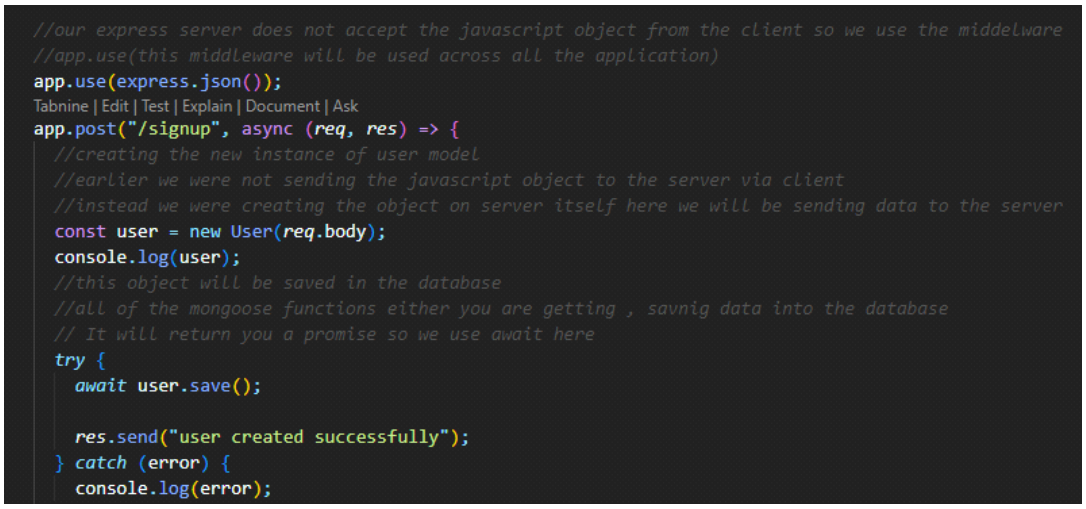
```
POST  http://localhost:3000/signup
Status: 200 OK   Size: 25 Bytes   Time: 218 ms
user created successfully
```

Here we get the data we have send into the database



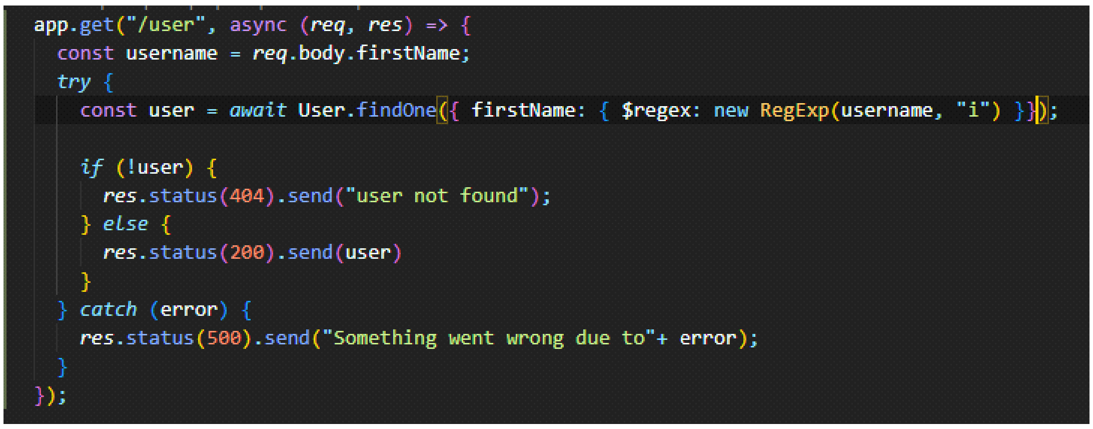
```
_id: ObjectId('675fd154ba1b7b4f0553e72e')
firstName : "Sam"
lastName : "habib"
age : 20
gender : "male"
email : "sam@gmail.com"
password : "alpha123"
__v : 0
```

How we send the json data via client dynamically?



```
//our express server does not accept the javascript object from the client so we use the middelware
//app.use(this middleware will be used across all the application)
app.use(express.json());
Tabnine | Edit | Test | Explain | Document | Ask
app.post("/signup", async (req, res) => {
  //creating the new instance of user model
  //earlier we were not sending the javascript object to the server via client
  //instead we were creating the object on server itself here we will be sending data to the server
  const user = new User(req.body);
  console.log(user);
  //this object will be saved in the database
  //all of the mongoose functions either you are getting , savnig data into the database
  // It will return you a promise so we use await here
  try {
    await user.save();

    res.send("user created successfully");
  } catch (error) {
    console.log(error);
```

How to find the entity from the database ?

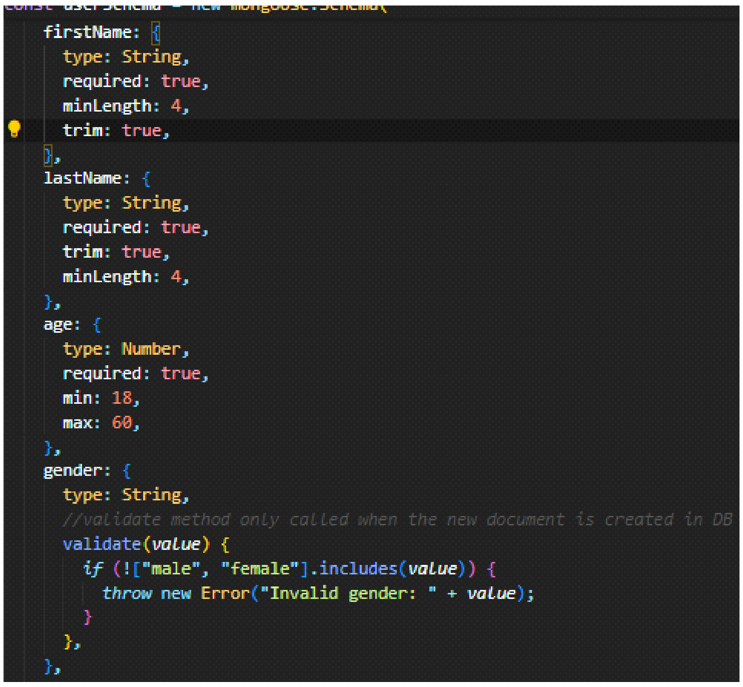


```
app.get("/user", async (req, res) => {
  const username = req.body.firstName;
  try {
    const user = await User.findOne({ firstName: { $regex: new RegExp(username, "i") }});

    if (!user) {
      res.status(404).send("user not found");
    } else {
      res.status(200).send(user)
    }
  } catch (error) {
    res.status(500).send("Something went wrong due to"+ error);
  }
});
```

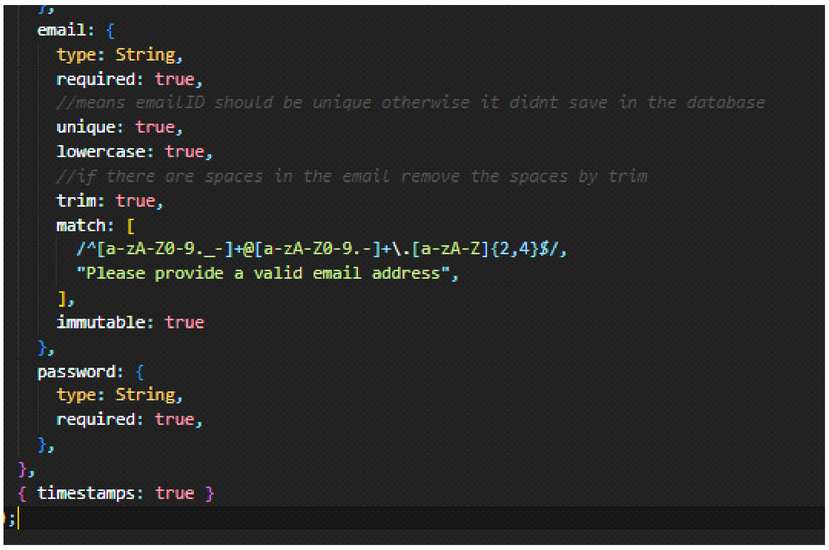
There are multiple functions which mongoose provide findOne ,find , and lot more we can check out from it official documentation.

How we apply the schema validation?

Here are some validation I did in schema

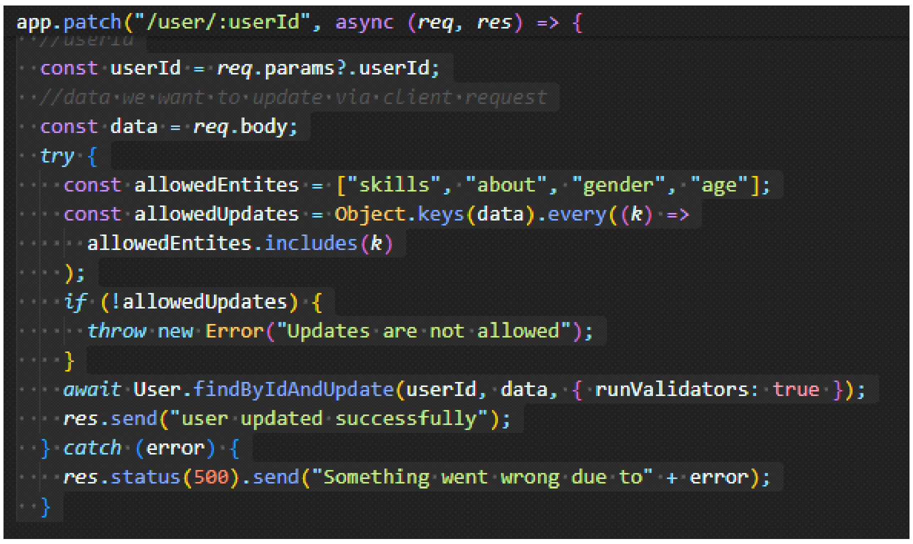


```
  firstName: {
    type: String,
    required: true,
    minLength: 4,
    trim: true,
  },
  lastName: {
    type: String,
    required: true,
    trim: true,
    minLength: 4,
  },
  age: {
    type: Number,
    required: true,
    min: 18,
    max: 60,
  },
  gender: {
    type: String,
    //validate method only called when the new document is created in DB
    validate(value) {
      if (!["male", "female"].includes(value)) {
        throw new Error("Invalid gender: " + value);
      }
    },
  },
```



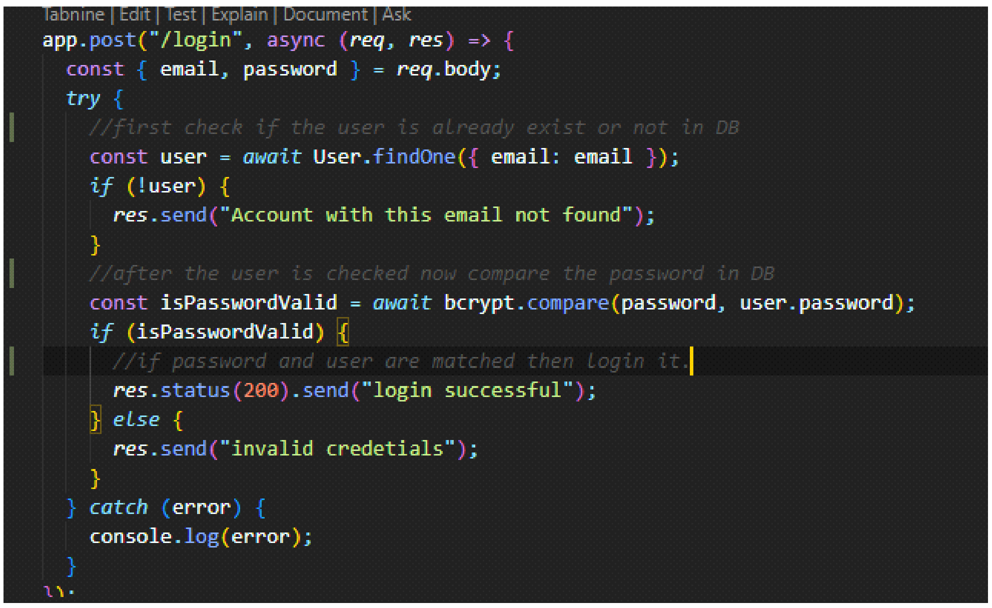
```
  email: {
    type: String,
    required: true,
    //means emailID should be unique otherwise it didnt save in the database
    unique: true,
    lowercase: true,
    //if there are spaces in the email remove the spaces by trim
    trim: true,
    match: [
      /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/,
      "Please provide a valid email address",
    ],
    immutable: true
  },
  password: {
    type: String,
    required: true,
  },
},
{ timestamps: true }
);
```

Adding Api level sanitization



```
app.patch("/user/:userId", async (req, res) => {
  //userId
  const userId = req.params?.userId;
  //data we want to update via client request
  const data = req.body;
  try {
    const allowedEntites = ["skills", "about", "gender", "age"];
    const allowedUpdates = Object.keys(data).every((k) =>
      allowedEntites.includes(k)
    );
    if (!allowedUpdates) {
      throw new Error("Updates are not allowed");
    }
    await User.findByIdAndUpdate(userId, data, { runValidators: true });
    res.send("user updated successfully");
  } catch (error) {
    res.status(500).send("Something went wrong due to" + error);
  }
```

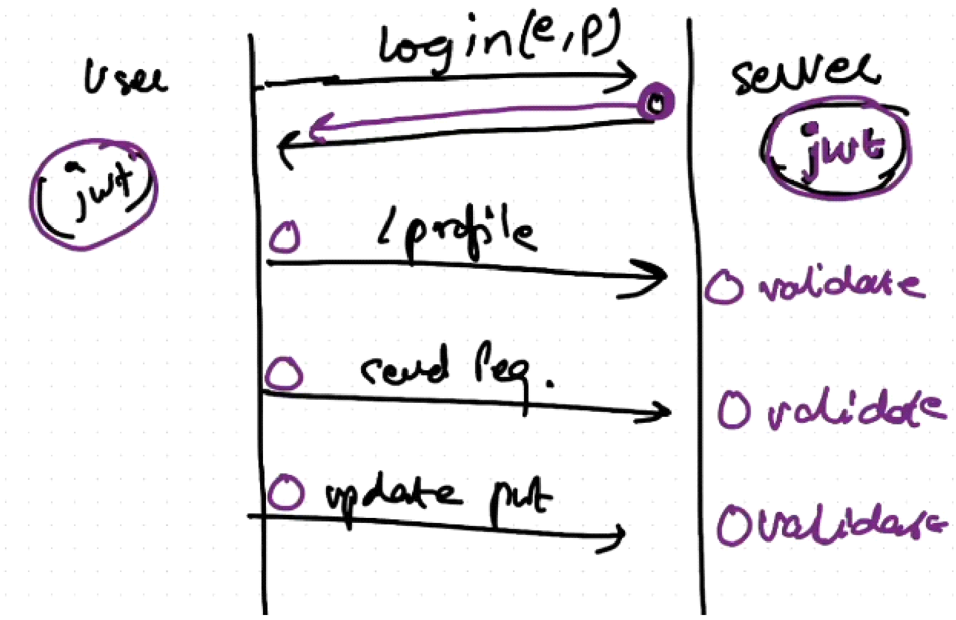
How to build the login API



```
app.post("/login", async (req, res) => {
  const { email, password } = req.body;
  try {
    //first check if the user is already exist or not in DB
    const user = await User.findOne({ email: email });
    if (!user) {
      res.send("Account with this email not found");
    }
    //after the user is checked now compare the password in DB
    const isPasswordValid = await bcrypt.compare(password, user.password);
    if (isPasswordValid) {
      //if password and user are matched then login it.
      res.status(200).send("login successful");
    } else {
      res.send("invalid credetials");
    }
  } catch (error) {
    console.log(error);
  }
});
```

Suppose when the user (client) from the browser makes the request to the server the server receive that request and response with data or message and close the connection each time client server connection is managed by tcp/ip protocol .

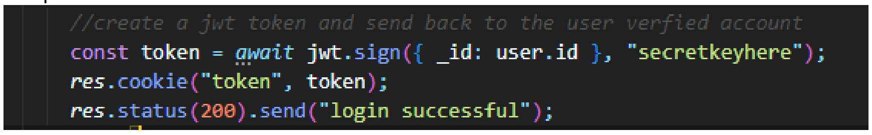
Suppose when the user want to login the profile it sends the request and server checks the credentials and login the user with some additional token within response this token is stored in a client browser and each time when the user want to update , delete or any task from it profile it is send to server and   it validates weather the user is authorized to do that or not.



Now how does this token actually stores in the browser and send back to server again and again so to tackle this solution came into the picture called cookies.

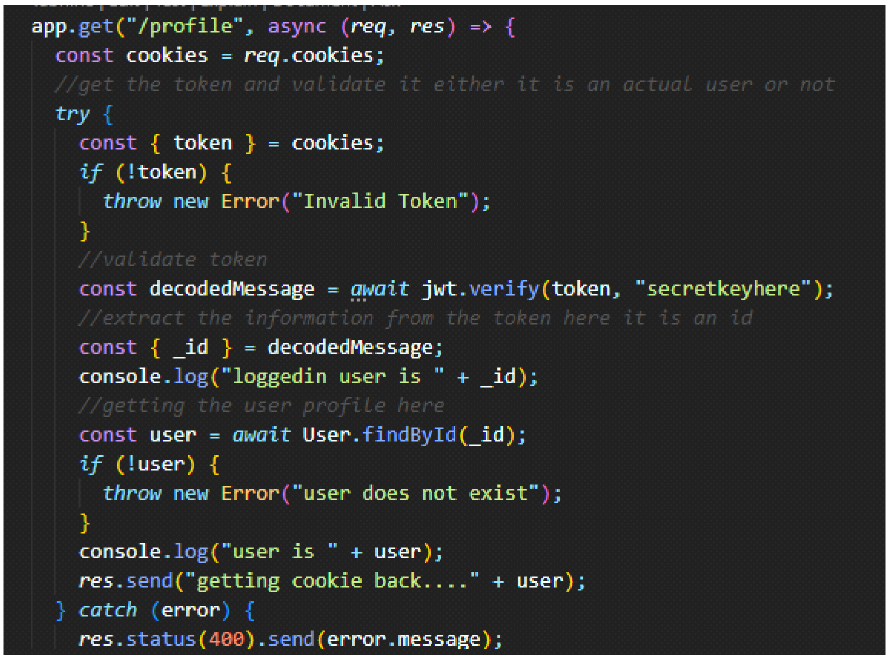
To read the cookie we need a middleware called cookie parser .

When the user want login we will create a token inside the /login api after the successful email id and password



```
//create a jwt token and send back to the user verfied account
const token = await jwt.sign({ _id: user.id }, "secretkeyhere");
res.cookie("token", token);
res.status(200).send("login successful");
```

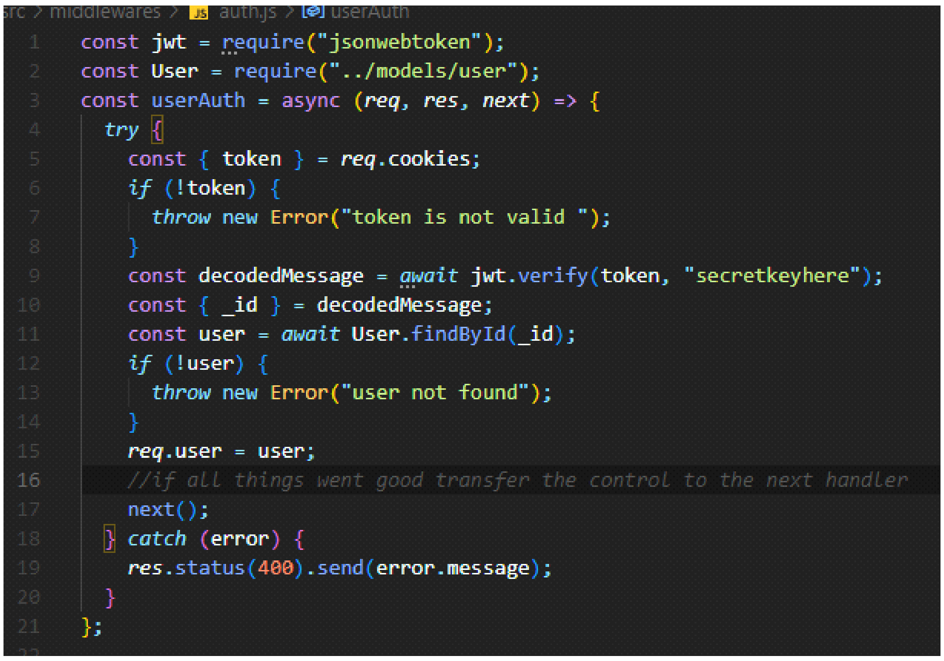
Here we get the profile of that user



```
app.get("/profile", async (req, res) => {
  const cookies = req.cookies;
  //get the token and validate it either it is an actual user or not
  try {
    const { token } = cookies;
    if (!token) {
      throw new Error("Invalid Token");
    }
    //validate token
    const decodedMessage = await jwt.verify(token, "secretkeyhere");
    //extract the information from the token here it is an id
    const { _id } = decodedMessage;
    console.log("loggedin user is " + _id);
    //getting the user profile here
    const user = await User.findById(_id);
    if (!user) {
      throw new Error("user does not exist");
    }
    console.log("user is " + user);
    res.send("getting cookie back...." + user);
  } catch (error) {
    res.status(400).send(error.message);
```

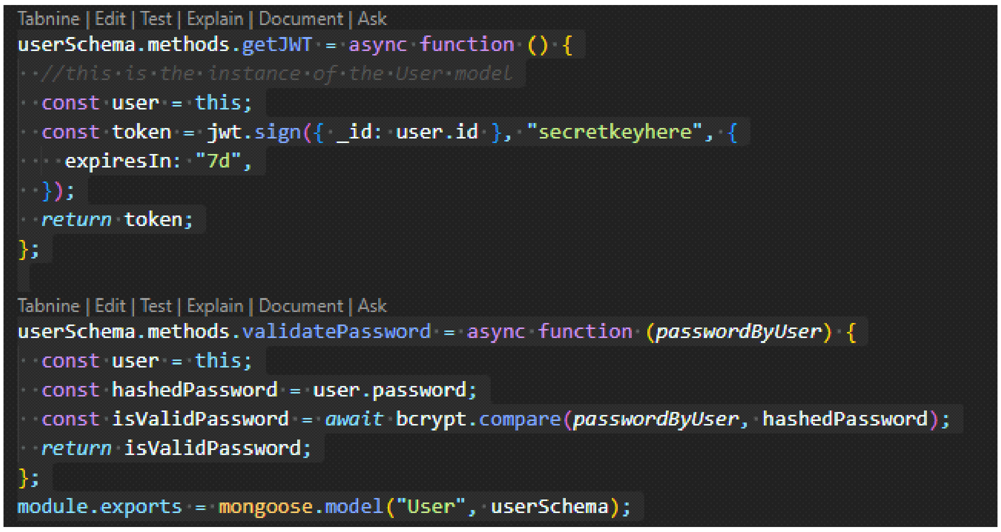
Hers how can we apply the cookie and jwt mechanisms into our app .

Creating the custom middleware for authentication of user



```
const jwt = require("jsonwebtoken");
const User = require("../models/user");
const userAuth = async (req, res, next) => {
  try {
    const { token } = req.cookies;
    if (!token) {
      throw new Error("token is not valid ");
    }
    const decodedMessage = await jwt.verify(token, "secretkeyhere");
    const { _id } = decodedMessage;
    const user = await User.findById(_id);
    if (!user) {
      throw new Error("user not found");
    }
    req.user = user;
    //if all things went good transfer the control to the next handler
    next();
  } catch (error) {
    res.status(400).send(error.message);
  }
};
```

Creating a custom schema methods for password validation and token verification as it is considered the good practice


Tabnine | Edit | Test | Explain | Document | Ask

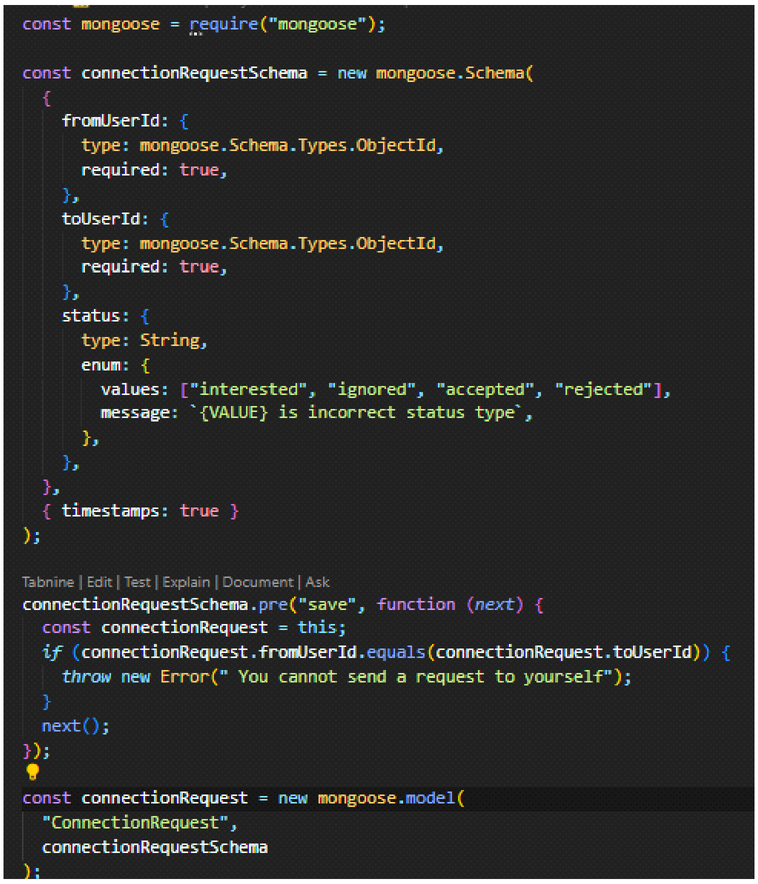
```
userSchema.methods.getJWT = async function () {
  //this is the instance of the User model
  const user = this;
  const token = jwt.sign({ _id: user.id }, "secretkeyhere", {
    expiresIn: "7d",
  });
  return token;
};
```

Tabnine | Edit | Test | Explain | Document | Ask

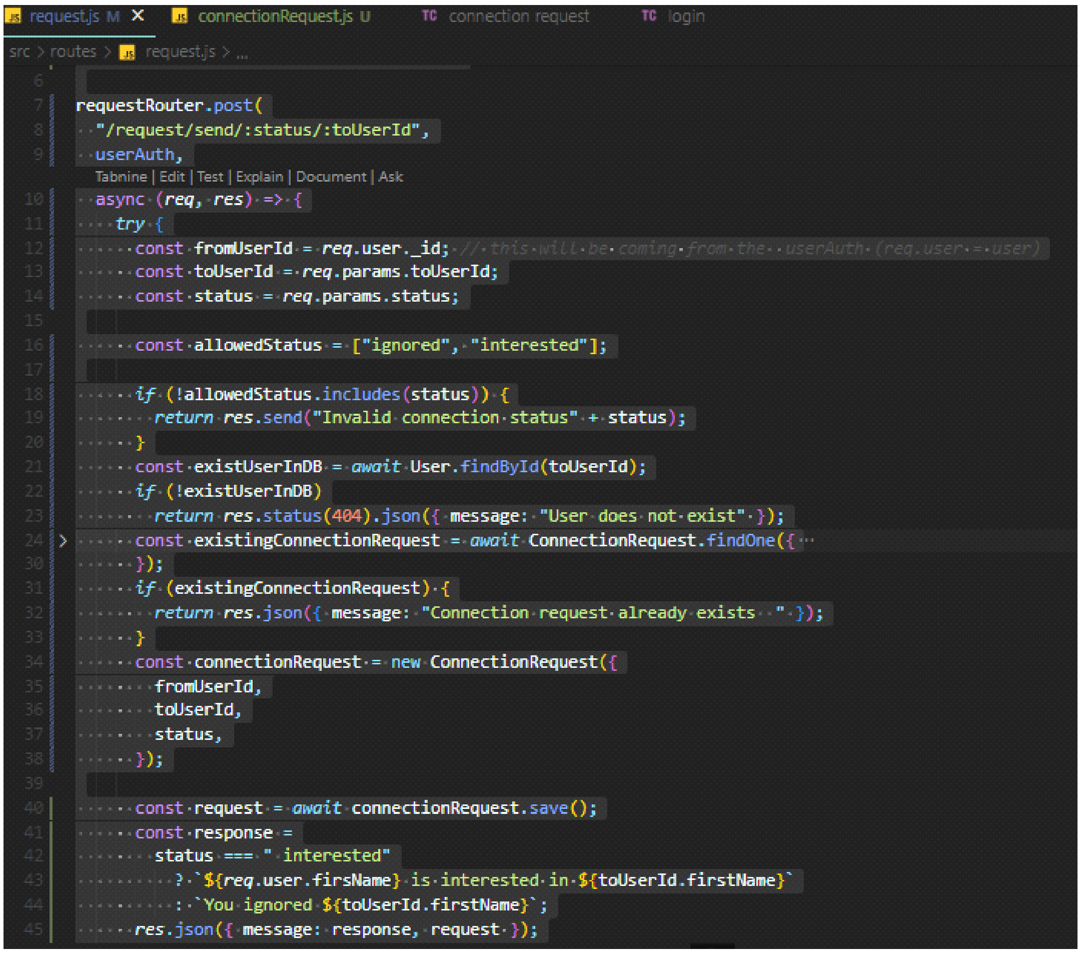
```
userSchema.methods.validatePassword = async function (passwordByUser) {
  const user = this;
  const hashedPassword = user.password;
  const isValidPassword = await bcrypt.compare(passwordByUser, hashedPassword);
  return isValidPassword;
};
module.exports = mongoose.model("User", userSchema);
```

How to send the connection request either ignored or interested in first create an schema

1- basically id is required to whom to send the request
2- Both id is required in the request we make
3- Validate the status using the num in schema

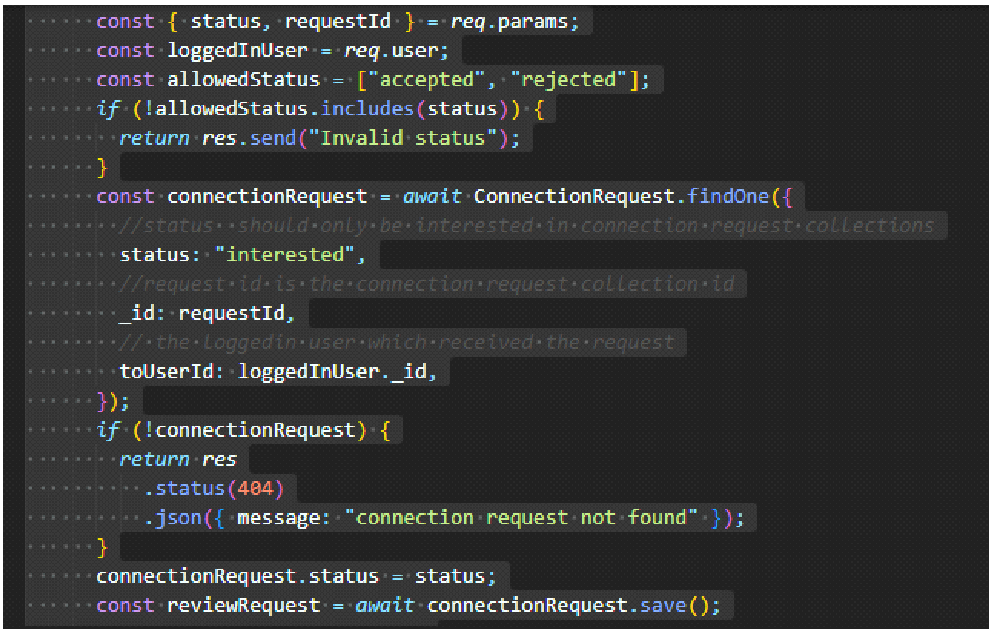


```js
const mongoose = require("mongoose");

const connectionRequestSchema = new mongoose.Schema(
  {
    fromUserId: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
    },
    toUserId: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
    },
    status: {
      type: String,
      enum: {
        values: ["interested", "ignored", "accepted", "rejected"],
        message: `{VALUE} is incorrect status type`,
      },
    },
  },
  { timestamps: true }
);

connectionRequestSchema.pre("save", function (next) {
  const connectionRequest = this;
  if (connectionRequest.fromUserId.equals(connectionRequest.toUserId)) {
    throw new Error(" You cannot send a request to yourself");
  }
  next();
});

const connectionRequest = new mongoose.model(
  "ConnectionRequest",
  connectionRequestSchema
);
```


request.js M ×
connectionRequest.js U
TC connection request
TC login
src > routes > request.js > ...
Tabnine | Edit | Test | Explain | Document | Ask

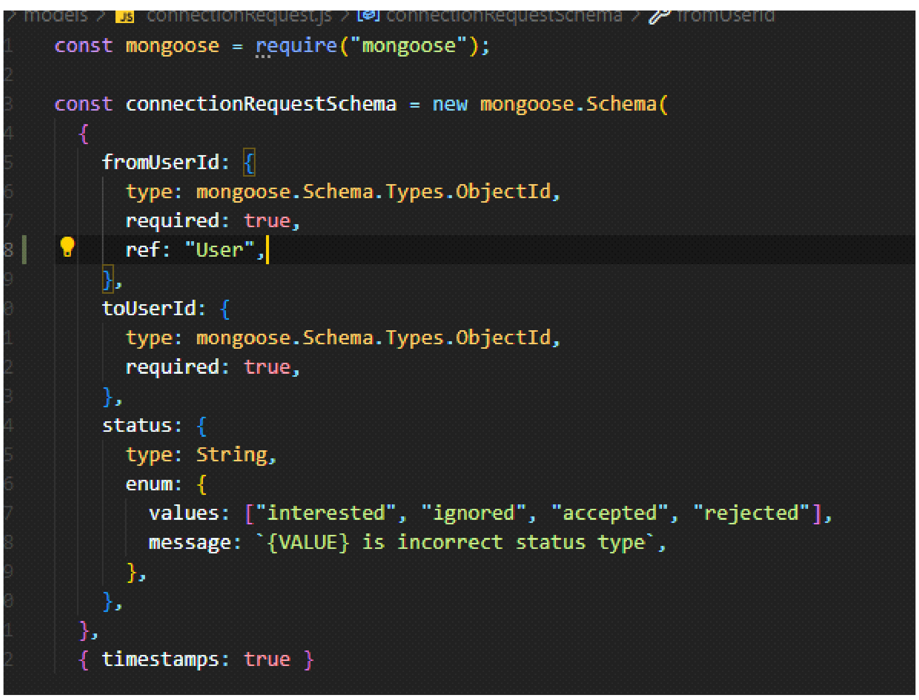
```js

requestRouter.post(
  "/request/send/:status/:toUserId",
  userAuth,
  async (req, res) => {
    try {
      const fromUserId = req.user._id; // this will be coming from the  userAuth (req.user = user)
      const toUserId = req.params.toUserId;
      const status = req.params.status;

      const allowedStatus = ["ignored", "interested"];

      if (!allowedStatus.includes(status)) {
        return res.send("Invalid connection status" + status);
      }
      const existUserInDB = await User.findById(toUserId);
      if (!existUserInDB)
        return res.status(404).json({ message: "User does not exist" });
      const existingConnectionRequest = await ConnectionRequest.findOne({ ···
      });
      if (existingConnectionRequest) {
        return res.json({ message: "Connection request already exists  " });
      }
      const connectionRequest = new ConnectionRequest({
        fromUserId,
        toUserId,
        status,
      });

      const request = await connectionRequest.save();
      const response =
        status === " interested"
          ? `${req.user.firsName} is interested in ${toUserId.firstName}`
          : `You ignored ${toUserId.firstName}`;
      res.json({ message: response, request });
```

Here is how we review the api for either to accept it or reject it

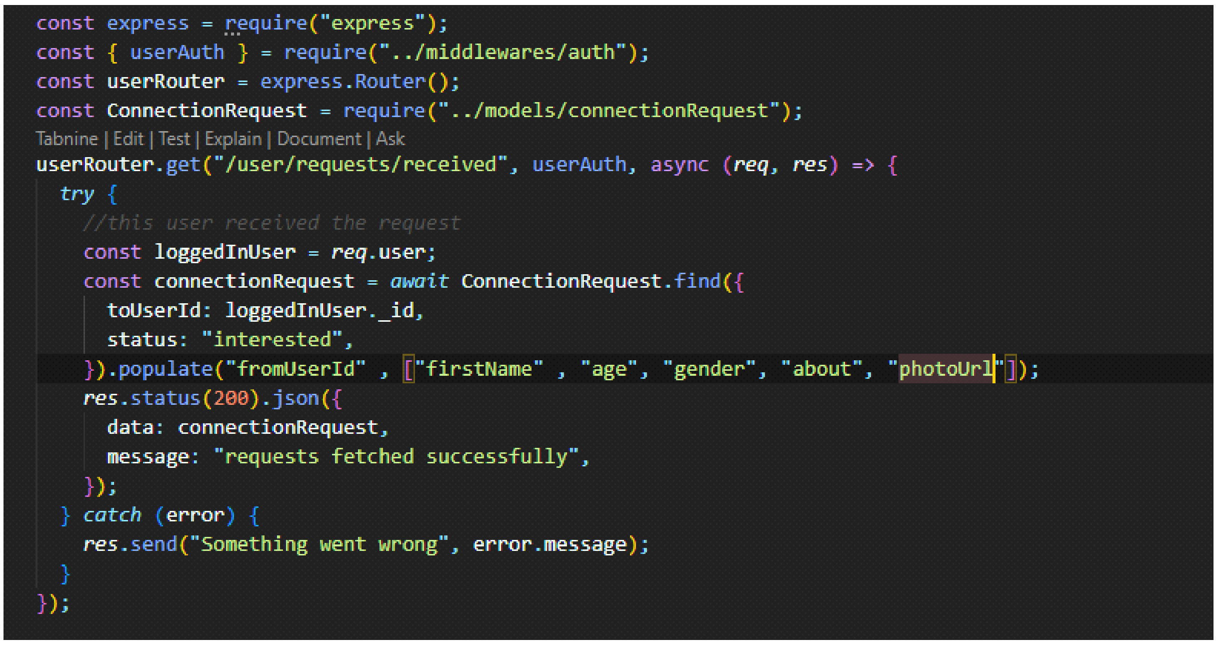

```
      const { status, requestId } = req.params;
      const loggedInUser = req.user;
      const allowedStatus = ["accepted", "rejected"];
      if (!allowedStatus.includes(status)) {
        return res.send("Invalid status");
      }
      const connectionRequest = await ConnectionRequest.findOne({
        //status  should only be interested in connection request collections
        status: "interested",
        //request id is the connection request collection id
        _id: requestId,
        // the loggedin user which received the request
        toUserId: loggedInUser._id,
      });
      if (!connectionRequest) {
        return res
          .status(404)
          .json({ message: "connection request not found" });
      }
      connectionRequest.status = status;
      const reviewRequest = await connectionRequest.save();
```

How do we make the relation between the collection in db?

Suppose we have there are two collections users and connections in this case the user which have sent the request to another user is present in the users collection and the connection sent stored in connections collection so when the user logged in and see the request I have received from xyz user so it must be shown to user which user sent you a connection request basically all the information of that user. In connections collection I am storing the id of both sending user and receiving user so how we get the data of user based on their ID's which are stored in users collection so the answer is we will be refrencing the Users in the connections collection schema model. And we will receive the specific information of the user who send the request by applying the populate method in response. Heres the example

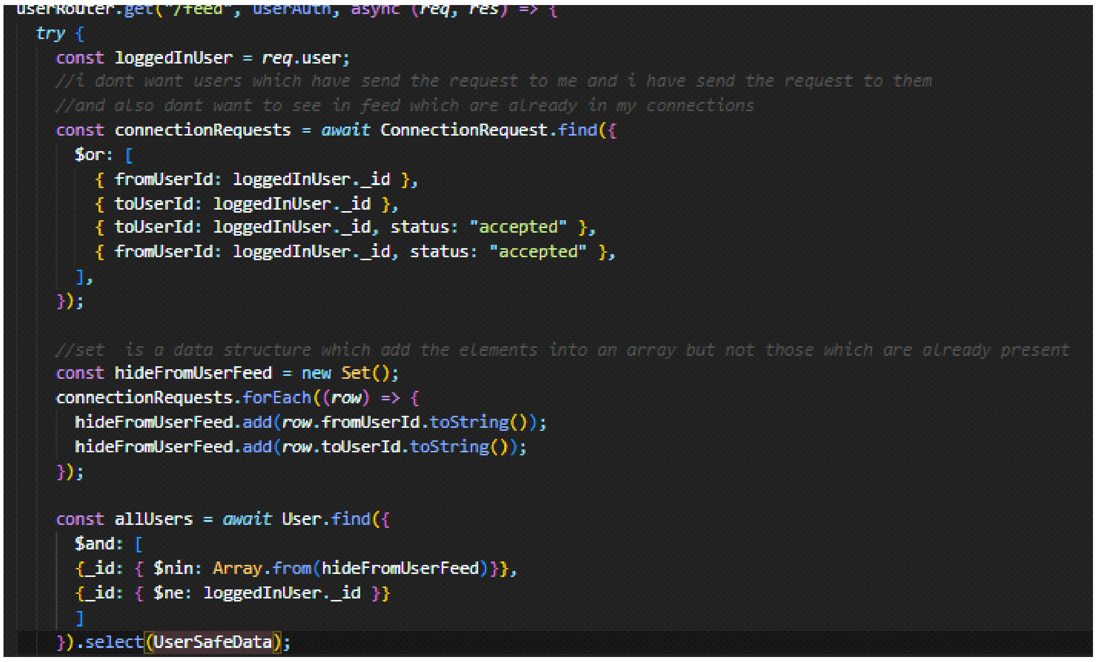

models > connectionRequest.js > connectionRequestSchema > fromUserId

```
const mongoose = require("mongoose");

const connectionRequestSchema = new mongoose.Schema(
  {
    fromUserId: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
      ref: "User",
    },
    toUserId: {
      type: mongoose.Schema.Types.ObjectId,
      required: true,
    },
    status: {
      type: String,
      enum: {
        values: ["interested", "ignored", "accepted", "rejected"],
        message: `{VALUE} is incorrect status type`,
      },
    },
  },
  { timestamps: true }
```



```
const express = require("express");
const { userAuth } = require("../middlewares/auth");
const userRouter = express.Router();
const ConnectionRequest = require("../models/connectionRequest");
Tabnine | Edit | Test | Explain | Document | Ask
userRouter.get("/user/requests/received", userAuth, async (req, res) => {
  try {
    //this user received the request
    const loggedInUser = req.user;
    const connectionRequest = await ConnectionRequest.find({
      toUserId: loggedInUser._id,
      status: "interested",
    }).populate("fromUserId" , ["firstName" , "age", "gender", "about", "photoUrl"]);
    res.status(200).json({
      data: connectionRequest,
      message: "requests fetched successfully",
    });
  } catch (error) {
    res.send("Something went wrong", error.message);
  }
});
```

How we build the feed api in this case ?



```
userRouter.get("/feed", userAuth, async (req, res) => {
  try {
    const loggedInUser = req.user;
    //i dont want users which have send the request to me and i have send the request to them
    //and also dont want to see in feed which are already in my connections
    const connectionRequests = await ConnectionRequest.find({
      $or: [
        { fromUserId: loggedInUser._id },
        { toUserId: loggedInUser._id },
        { toUserId: loggedInUser._id, status: "accepted" },
        { fromUserId: loggedInUser._id, status: "accepted" },
      ],
    });

    //set  is a data structure which add the elements into an array but not those which are already present
    const hideFromUserFeed = new Set();
    connectionRequests.forEach((row) => {
      hideFromUserFeed.add(row.fromUserId.toString());
      hideFromUserFeed.add(row.toUserId.toString());
    });

    const allUsers = await User.find({
      $and: [
      {_id: { $nin: Array.from(hideFromUserFeed)}},
      {_id: { $ne: loggedInUser._id }}
      ]
    }).select(UserSafeData);
```

The above example allUsers I will get the users which are not from the connectionRequest object.

```
const page = parseInt(req.query.page) || 1;
    let limit = parseInt(req.query.limit) || 10;
    limit = limit > 50 ? 50 : limit;
    const skip = (page - 1) * limit;


.select(UserSafeData)
      .skip(skip)
      .limit(limit);
```