Class 8 - Authorization and conditional flow of data

Authorization

1. Lets talk about creating User
   a. We can signup and create a new user
   b. Admin can create entries
2. So basically two personas have access to do this action
3. Scaler’s admin for instance might have access to lot of learner’s data like email, scores, psp. Etc but one user might not have access to lot of other user’s information
4. Basically the idea is to identify different actions and then figure out the personas who have the right access to them
5. We will begin by adding a Role in User model
6. We will implement Two types of authorization mechanism
   a. Where we will restrict admins only to have access to certain actions
   b. Some other roles for these actions
   c. So the central idea of the discussion is that our authorization mechanism should be robust as well as flexible to adapt to business needs

Implementing isAdmin kind of a flow

1. Add a role attribute in the userSchema

   a. Default value of user

   ```
   otpExpiry:Date,
     role:{
       type: String,
       default: "user"
     },
   ```

2. We had get all users route
   a. We want to now restrict any user to access this route
   b. Who should have access to all the user data - admin
   c. We will place an authentication middleware to make sure user is authenticated
   d. Then we will place a middleware to authorize only admins to get access to the data in the route
3. Add this in userRouter

   ```
   userRouter.get('/allUsers',protectRoute, isAdmin,
   getAllUsers)
   ```

Now that we have lot of authentication related code, let us move all this into a separate file

Go over all the existing code on github to get an idea where all the coed is

Create a file authController.js under controllers folder.

1. All the authentication related code will be moved to this folder like signup, login forget and reset password
2.
   i. imports

```
const jwt = require("jsonwebtoken");
const UserModel = require("../models/userModel");
const { sendEmailHelper } = require("../nodemailer");
const User = require("../models/userModel");
const { SECRET_KEY } = process.env;
```

   ii. otpGenerator method

   iii. signupHandler

```
const signupHandler = async function (req, res) {
 try {
   // add it to the db
   const userObject = req.body;
   //   data -> req.body
   let newUser = await UserModel.create(userObject);
   // send a response
   res.status(201).json({
     message: "user created successfully",
     user: newUser,
     status: "success",
   });
 } catch (err) {
   console.log(err);
   res.status(500).json({
     message: err.message,
     status: "success",
   });
 }
};
```

iv. loginHandler

v. forgetPassword from userController

```
const forgetPassword = async (req, res) => {
   // user sends their email
   // verify that the email exists
   // generate a token
   // send the token to the user's email
   try {
     const { email } = req.body;
     const user = await User.findOne({ email });
     console.log("user", user)
     if (!user) {
       res.status(404).json({
         message: "error",
         data: "User not found",
       });
     } else {
       // generate a token
       const token = otpGenerator();
       user.token = token.toString();
       user.otpExpiry = Date.now() + 1000*60*60; // 60
minutes
       console.log("ipdated user", user)
       await user.save();
       sendEmailHelper(token, email)
         .then(() => {
           console.log("Email is sent");
         })
         .catch((err) => {
           console.log(err);
         });
         res.status(200).json({
             message: "success",
             data: "Email is sent",
             id: user._id,
             token: user.token
         })
        // save the token in the user's document
       // send the token to the user's email
     }
   } catch (err) {
     res.status(500).json({
       message: "error",
       data: err.message,
     });
   }
 };
```

vi. resetPassword

```
const resetPassword = async (req, res) => {
   // user sends their token and new password
   // req has user id in the params
   // verify that the token is valid
   // update the user's password
```

```
try{
  const { token, password, email } = req.body;
  const { userId } = req.params;
  // logic
  /**
   * 1. find the user with the id
   * 2. check if the token is valid
   * 3. check if the token is expired
   * 4. upate the password
   */
  console.log("email", email)
 //  const user = await User.findOne({email});
 // convert string id to ObjectId
 console.log("id", userId)
 const user = await User.findById(userId);
   console.log("user", user)
  if(!user){
    res.status(404).json({
      message: "error",
      data: "User not found",
    });
  }else {
     if(user.token !== token){
       res.status(400).json({
         message: "error",
         data: "Invalid token",
       });
     }else {
       if(user.otpExpiry < Date.now()){
         res.status(400).json({
           message: "error",
           data: "Token is expired",
          });
        }else {
          user.password = password;
          user.token = null;
          user.otpExpiry = null;
          await user.save();
          res.status(200).json({
            message: "success",
            data: "Password is updated",
          });
        }
      }
   }
  }catch(err){
  res.status(500).json({
    message: "error",
    data: err.message,
  });
 }

};
```

4. We will implement a protect route method here as well

```
const protectRoute = async (req, res, next) => {
   // get token from cookies
   // verify the token
   // get user from database
   // if user exists then allow next()
 }
```

```
const protectRoute = (req, res, next) => {
   try{
       const token = req.cookies.token
       const decoded = jwt.verify(token,
process.env.SECRET_KEY);
       if(decoded){
           const userId = decoded.id;
           req.userId = userId;
           next()
       }


   }catch(err){
       res.status(500).json({
           status: "failed to authenticate",
           message: err.message
       })
   }

}
```

4. isAdmin check would be simple

```
const isAdmin = async (req, res, next) => {
   // check if user is admin
   const userId = req.userId;
   const user = await User.findById(userId);
   if(user.role === "admin"){
       next()
   }else {
       res.status(401).json({
           status: "failed to authenticate",
           message: "You are not authorized to access this
route"
       })
   }
}
```

5. Import the methods in userRouter

```
const {signupHandler, loginHandler, isAdmin, protectRoute}
= require("../controllers/authController");
```

6. Add the routes for signup and login

```
userRouter.post("/signup", checkInput, signupHandler);
userRouter.post("/login", checkInput, loginHandler);
```

7. In postman , create two user, one with role = admin and one without role attribute
8. Now login - `localhost:3000/api/users/login`
9. Make sure to have cookie parser in your app.js if missing

```
const cookieParser = require("cookie-parser");
app.use(cookieParser());
```

Creation of Auth Router

1. Create a new router for authentication which will have signup , login, forger and reset password

```
const express = require("express");
const authRouter = express.Router();

const { signupHandler, loginHandler, resetPassword,
forgetPassword } =
require("../controllers/authController");
const { checkInput } = require('../utils/crudFactory');
/** Routes for user */
authRouter.post("/signup", checkInput, signupHandler);
authRouter.post("/login", checkInput, loginHandler);
// forget and reset password routes
authRouter.patch("/forgetPassword", forgetPassword);
authRouter.patch("/resetPassword/:userId", resetPassword);

module.exports = authRouter;
```

2. Add the auth router in the app.js

```
app.use("/api/auth",authRouter);
```

3. One additional thing we can do is for all the routes of user , we want to use the protect handler

```
userRouter.use(protectRoute);
```

Demo accessing getAllUser route with both the users

Check for cookie parser if token related error

Problem Statement

1. Lets say we want to introduce a new manager role in our project

2. We want for example our manager and admin to view all users but only admins to delete a user
3. Similarly there could be a seller role which allows a user to create a product and delete a product they have created
4. First let us have a pre hook for userModel to validate and restrict the roles

```
const validRoles = ["admin", "user", "seller"];
 userSchema.pre("save", function(){
   this.confirmPassword = undefined;
   // checking for roles
   if(this.role){
     const isValid = validRoles.includes(this.role);
     if(!isValid){
       next(new Error("User can either be admin, user or
seller"))
     }else {
       next()
     }

   }else {
     this.role = "user";
     next()
   }

 })
```

5. Now in the authController we had created a static check for admin role

6. let us create a generic function which will accept allowed roles and check user against those roles

```
const isAuthorized = function (allowedRoles) {
 return async function (req, res, next) {
   // check if user is admin
   const userId = req.userId;
   console.log("userId", userId);
   const user = await User.findById(userId);
   if (allowedRoles.includes(user.role)) {
     next();
   } else {
     res.status(401).json({
       status: "failed to authenticate",
       message: "You are not authorized to access this
route",
     });
   }
 };
};
```

7. This is similar to what we did for admin. Instead of making it specific to a role , we will pass the set of roles and create a closure over the roles
8. In the productRouter file, let us use these control check middlewares

```
const authorizedProductRoles = ["admin", "ceo", "sales"];

productRouter.post("/", checkInput, protectRoute,
isAuthorized(authorizedProductRoles),createProduct);
```

9. For delete operation as well, let us add these checks

```
productRouter.delete("/:id", protectRoute,
isAuthorized(authorizedProductRoles),deleteProductById);
```

10. Check creation of products with different roles

Logout

1. Add a route on authRouter

```
authRouter.get("/logout", logoutHandler);
```

2. Logout handler in authController

```
const logoutHandler = (req, res) => {
   // clear the cookie
   res.clearCookie("token");
   res.status(200).json({
     status: "success",
     message: "User is logged out",
   });
}
```

3. Client-Side Considerations: Ensure that the client application properly handles the logout process. This typically involves removing any stored session information (like tokens stored in local storage, if applicable) and updating the UI to reflect the logged-out state.
4. In postman, logout and try to access a route