NAMASTE NODE.JS SEASON 2

Episode - 10
Hand Written Notes

-By Shanmuga Priya

www.linkedin.com/in/shanmuga-priya-e-tech2

Episode - 10

Authentication, JWT & Cookies

1) Why we need Authentication?

→ whenever a client req some data from the server TCP/IP Protocol is made between client & server and server checks / validate the user. If the user is a authorized (logged in) the server sends back the data to client and connection is closed

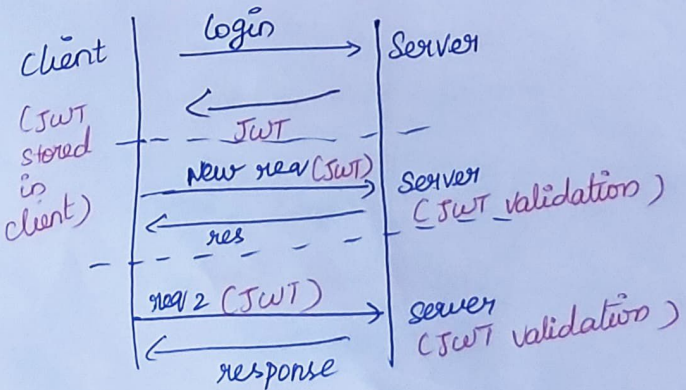
→ This Process of making an connection, validating the user, sending res back & closing the connection happens for all the requests made by a user.

2) How server validate the User who is making a request?

→ when the user logs in initially the server creates a JWT token and sends back to client along with response

→ The JWT token is stored on client side & everytime when the user makes an API call it sends back the token from client to server.

→ The server then validates the incoming Token whether it is malformed (or) not.



3) Where the JWT tokens get stored?

→ The JWT token gets stored in cookies in browser.

4) When the cookies will not work?

→ whenever the cookies (or) JWT token gets expired.

5) How to create a cookies & how to access it?

Creating cookie:

→ we can create a new cookie by res.cookie() which is given to us by Express.

→ res.cookie(Name, value)

→ Name of our cookie, value is the JWT token created.

Getting access to cookie:

→ we can get the cookie using req.cookies but we need to parse the cookie first inorder to read its value.

→ For that we use the middleware called Cookie-Parser

npm i cookie-parser.

app.use(cookieParser())

→ once we got the cookie we extract the JWT token from it & use for validation & send res back.

6) What is JWT?

→ JWT stands for JSON webToken is a way of securely sharing information between client and server.

→ It is commonly used for authentication and authorization in microservices.

7) what is the structure of JWT?

→ The JWT token consits of three parts seperated by dots (.)

*) Header → contain metadata like Token type (JWT) and hashing algorithm used (HS256)

*) Payload → contains the data that we want to send like userID, role.

*) signature → It is used to validate the token whether some has changed (or) not. It is a combination of header, Payload and secret key we set.

8) How to create a JWT token?

→ To create a JWT token we first need to install the JWT library.

npm i jsonwebtoken

→ There is a fn called "sign()" which accepts the payload, secretkey which is used to create a JWT token.

eg:

```
app.post("/login", async (req, res) => {
    try {
        const {email, password} = req.body
        //checking user in DB
        const user = await User.findOne({email : email})

        if (!user){
            throw new Error("user not found")
        }
        // comparing Password
        const isPasswordValid = await bcrypt.compare(password,
                                user.password)

        if (isPassword valid){
        //creating token
        const token = await Jwt.sign({_id:user._id},
```

```
        secret key )
    // Adding token to cookie & send back to client along with response
    res.cookie ("token", token)
    res.send ("login successfull")
  } else {
      throw new Error ("Invalid credentials")
  }
 } catch (err) {
    res.status (400). send ("Error" + err.message)
 }
})
```

7) How to verify the JWT token on subsequent request?

→ To validate the incoming Jwt token on further request there is a function called "verify" which accepts the incoming token & secret key which is used when creating a token.

eg:

```
app.get ("/profile", async (req, res) => {
  try {
    const cookies = req.cookies
    const {token} = cookies
    if (!token) {
      throw new Error ("Invalid token")
    }
    // decoding/ validating the token to get the Payload back
    const decodedToken = await jwt.verify (token, secretkey)
    const {_id} = decodedToken
    // finding user profile based on id stored in token
    const user = await User.findById (_id)
```

→ getting the cookie & token from that cookie

```
if (!user){
    throw new Error("user does not exist")
}
res.send(user)
} catch(err){
    res.status(400).send("Error" + err.message)
}
})
```

10) How to create a Auth middleware?

-> we need to provide access to all the route only after logged in except signup & login route.

-> Inorder to validate JWT token for all the route we are creating a middleware which will validate the token & sends back the user corresponding to it

-> Then we pass the req handlers after that to handle the api request.

Creating middleware

eg.

```
const userAuth = async (req, res, next) => {
    try {
        //getting token & validating it
        const {token} = req.cookies
        if (!token){
            throw new Error("Invalid token")
        }
        const decodedToken = await jwt.verify(token, secretkey)
        const {_id} = decodedToken
        //getting user
        const user = await User.findById(_id)
        if (!user){
            throw new Error("user not found")
        }
        //sending user
        req.user = user
```

```
        next()
    } catch (err) {
        res.status(400).send("ERROR " + err.message)
    }
})
```

using Middleware:

```
app.get("/profile", userAuth, async (req, res) => { })
```

11) How to set a expiry time for the JWT token?

→ we can set the expiry time for the token when creating it by passing a "expiresIn" Prop as a 3rd argument to "sign" fn.

eg:
```
const token = await jwt.sign(data, secretKey, {
    expiresIn: '1h'
})
```

12) How to set a expiry time for the cookies?

→ Similarly we can set the expiry time for cookie when creating it by passing "expires" prop as 3rd argument to "res.cookie()"

eg:
```
res.cookie("token", token, { expires: new Date(Date.now
    + 8 * 3600000) })
```

13) What are Schema methods?

→ These are methods that are directly attached to Schema and are available for all the document that are created based on that Schema.

→ Arrow fn are not allowed for Schema methods as "this" keyword is undefined.

14) How to create Schema methods?

→ to create a Schema methods we can use "Schema.methods" fn to attach a fn to the schema.

eg: // userSchema.js

```
userSchema.methods.validatePassword = async function (pwd) {
    const isPasswordValid = await bcrypt.compare (pwd, this.password)
                                                   ↑ incoming pwd
    return isPasswordValid
}
```

this → this points to current user document (hashed pwd)

Usage of Schema Method:

```
const isPasswordValid = await user.validatePassword (password)
```

→ This Schema methods enable us to have a cleaner, modular and maintainable code.
