Implementing Authentication with Cookies, Node & Express

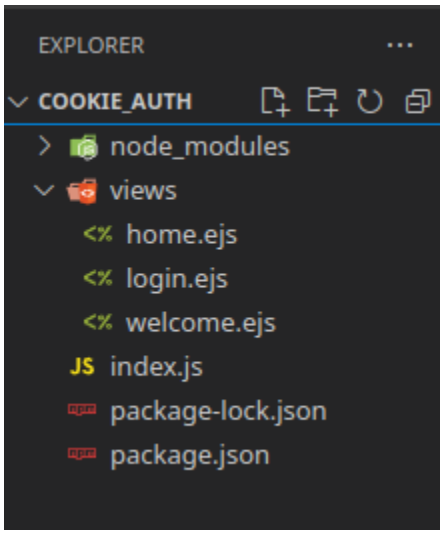
Initial Setup

In this project we will be learning how to create authentication using cookies, we will also learn how to use template engines like ejs and how to make secure requests using the Helmet package.

We will have to create an empty node project named `cookie_auth` in VS Code as discussed earlier.
Then we will start a couple of packages like ExpressJs, Cookie Parser and Helmet.
We can install multiple packages with a single line if we separate the name of the packages by space, like :

```
npm i express ejs cookie-parser helmet
```

Now create a `index.js` file which will be the starting point
Create a folder `views` which will have all the `.ejs` files, these files will be finally rendered as html files in the browser.
Create `home.ejs` , `login.ejs` for login form and `welcome.ejs` for greeting the user after login.
At this point the folder structure looks like this :



Starting with importing all the packages (`express path cookie-parser helmet`) in the index.js file.

Sometimes we can have a CORS error in the browser as it can block the secure requests, we are using the helmet package for that.

Create a basic expressjs server as discussed earlier.

Register the helmet() , cookieparser() , express.json(), express.urlencoded({extended:true}) middlewares with app.use() method.

Now we will have to set the view engine to ejs as :
`app.set("view engine", "ejs")` this will tell Express to use the ejs template engine to render views.

EJS is a template engine which we can use to provide dynamic data to render html instead of using static data, it allows us to embed JavaScript code into our HTML.

Now we have to set the directory from where we are going to get the pages, like :
`app.set("views", path.join(__dirname, 'views'))`
This will tell the template engine that we can get all the template “views” from the specified folder location.

At this point the indes.js file looks like this :

package.json | index.js | home.ejs | login.ejs | welcome.ejs

index.js > app.get('/') callback

```javascript
const express = require("express")
const helmet = require("helmet")
const cookieparser = require("cookie-parser")
const path = require("path")

const app = express();
const port = 3000;

app.use(helmet())
app.use(cookieparser())
app.use(express.json())
app.use(express.urlencoded({ extended: true }))

app.set("view engine", "ejs")

app.set("views", path.join(__dirname, 'views'))


0 references
app.get('/', (req, res) => {
    res.send("Home Page")
    res.end()
})

0 references
app.listen(port, () => {
    console.log("server running on port : ", port);
})
```

Setting up routes

Normally when we are using `send()` method we are using static data but here we have to use dynamic data so we will use `render()` method and send values to the ejs file and accordingly the content will be updated.

We can set up the / and /login routes like this :

```
app.get('/', (req, res) => {
    let username = req.cookies.username;
    return res.render("home", { username })
    // the object, i.e. 2nd param will be sent to the home.ejs
    file and accordingly the home.ejs file will be updated.
})

0 references
app.get('login', (req, res) => {
    let status = req.query.msg ? true : false;
    //depending on the status of login we are altering the html
    in the login.ejs file to be invalid or successful login.
    if (status === false) {
        return res.render("login", { error: "Invalid Details" })
    }
    else {
        return res.render("login")
    }
})
```

Similarly we will develop the `/welcome` , `/process_login` , `/logout` route, in the `/welcome` route we will just display a greeting message and pass the name of the user that we will get from the cookie data.

Then we will define the `/process_login` route where we will be receiving the form data from the client. And, finally we will delete the cookie to logout the user in `/logout` route.

At this point our `/welcome` & `/process_login` , `/logout` routes look like this :

```
JS index.js ×
JS index.js > app.post('/process_login') callback > validUserDetail
0 references
app.get("/welcome", (req, res) => {
    let username = req.cookies.username;
    return res.render('welcome', { username })
})
0 references
app.post('/process_login', (req, res) => {
    //post request on /process_login to get form data
    let { username, password } = req.body;

    let validUserDetail = {
        // sample data from which we will verify if the userdetails are correct
        username: "Barbarik",
        password: 'pswd0987'
    }
    if (username === validUserDetail.username && password === validUserDetail.password) {
        //i.e., the provided user details are correct => set the cookie
        res.cookie('username', username);
        return res.redirect('/welcome')
        // after saving data to cookie redirect to welcome page & from there get cookie for
        showing greeting
    }
    else {
        //if login not success
        return res.redirect("/login?msg=fail")
    }
})
```

```
0 references
app.get('/logout', (req, res) => {
    // to logout just delete the cookie from the client side
    res.clearCookie('username');
    // after deleting redirect to the login page
    return res.redirect('/login');
})

0 references
app.listen(port, () => {
    console.log("server running on port : ", port);
})
```

Remember that we are passing some values to the ejs files in many routes and we will learn how to make use of them.

Setting up the .ejs files

EJS stands for Embedded JavaScript, and it's a templating language that allows you to dynamically generate HTML pages on the server-side. EJS files have the extension ".ejs" and are used with server-side JavaScript frameworks such as Node.js or Express.

The basic idea behind EJS is that you write HTML code with placeholders for dynamic content, and then use JavaScript code to fill in those placeholders with actual content when the page is rendered. Let's take a look at an example to see how this works.

Any JS code portion has to be written inside `<%` and `%>` placeholders and if we are accessing a variable then we have to use `<%=` and `%>` this makes it very easier to embed JS inside html. The objects that are passed to the ejs file from the routes in the express app are available inside the `locals` object.

The `home.ejs` file

```
JS index.js    <% welcome.ejs    <% home.ejs ×
views > <% home.ejs > html
1   <html>
2
3   <head>
4       <title>Home</title>
5   </head>
6
7   <body>
8       <h4>Hello there, Welcome!</h4>
9       <% if(locals.username) { %>
10          <p>
11              You are logged in as <%= username %>
12          </p>
13          <a href="/logout">Logout</a>
14          <% } else{ %>
15              <a href="/login">Login</a>
16              <% } %>
17
18  </body>
19
20  </html>
```

The `login.ejs` file

```
JS index.js    <% home.ejs    <% login.ejs ×    <% welcome.ejs
views > <% login.ejs > html > body > ? > ? > form > br
1   <html>
2
3   <head>
4       <title>Login</title>
5   </head>
6
7   <body>
8       <h4>Login</h4>
9       <% if(locals.error) { %>
10          <p>
11              <%= error %>
12          </p>
13          <% } %>
14
15              <form action="/process_login" method="post">
16                  <input type="text" name='username'><br>
17                  <input type="password" name='password'><br>
18                  <input type='submit' value="Login">
19              </form>
20  </body>
21
22  </html>
```

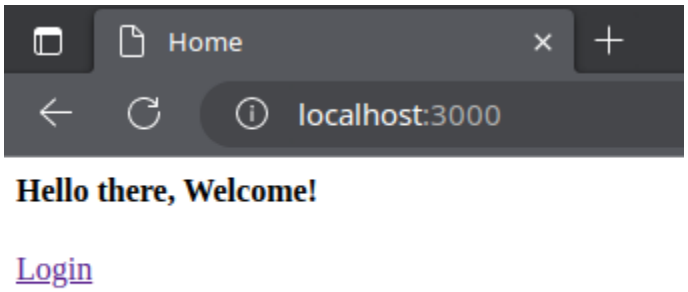
The login page will have a form and we can provide username and password to login.

The `welcome.ejs` file

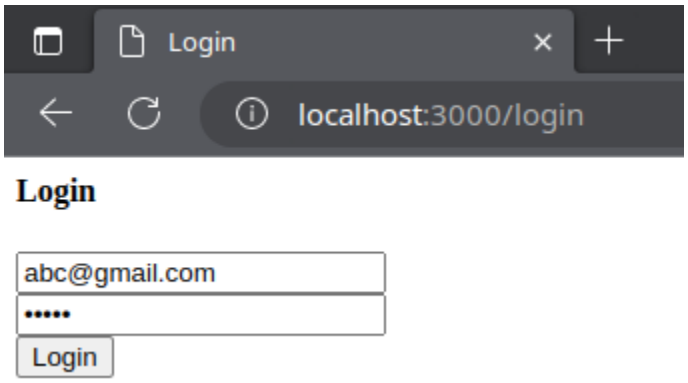
```
JS index.js    <% welcome.ejs ×    <% home.ejs
views > <% welcome.ejs > html
1   <html>
2
3   <head>
4       <title>Welcome</title>
5   </head>
6
7   <body>
8       <h1>Hi <%= username %>, Welcome </h1>
9       <a href="/logout">Logout</a>
10  </body>
11
12  </html>
```

This page displays a simple welcome page and a link to logout.

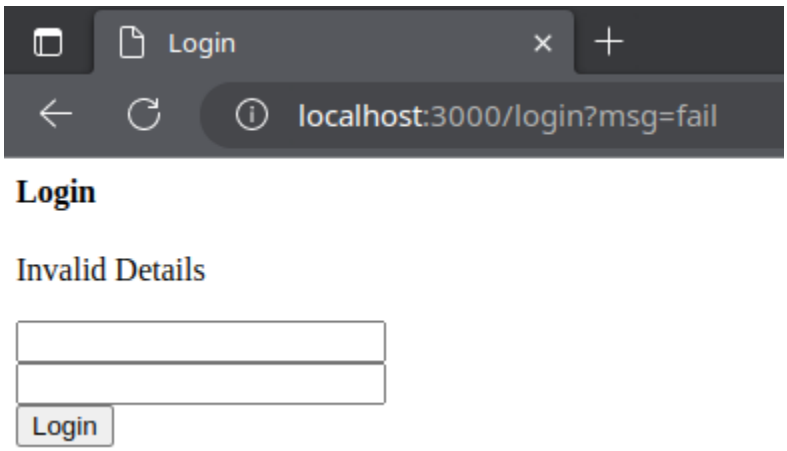
Now if you spin the server by node index.js, and visit to localhost:3000 route



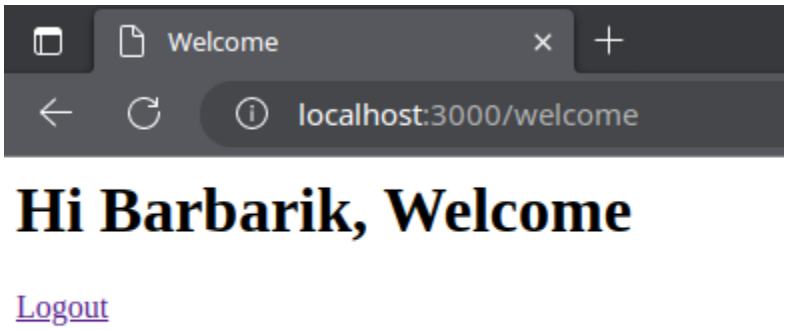
The home.ejs is rendered in html like this
Click the Login link for visiting the Login form



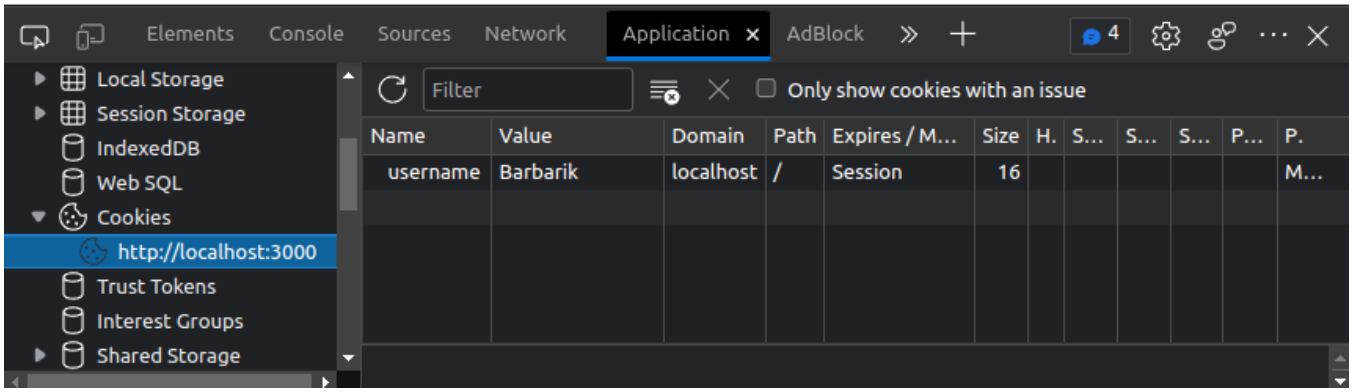
On providing some random credentials at the login page we get



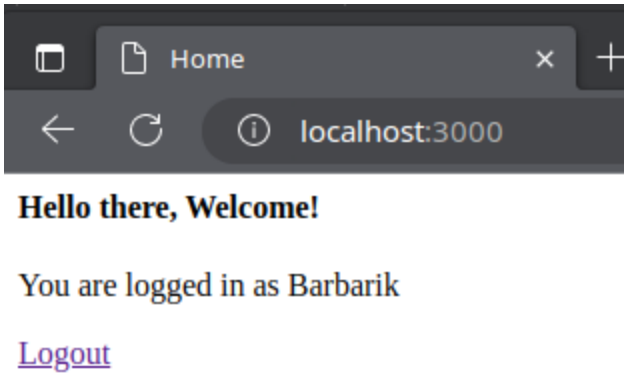
Now if we provide correct credentials as defined in the /process_login route



From here if you will look at the cookies that are present it will look like this :



Now at this point / looks like :



Now if the cookie is deleted then :