Backend Development

DA217A

Lab 4

1. Laboratory goals

In the previous lab you learned how to make a login. However, a login is useless unless you actually have access control. Meaning if anyone can view the pages without being logged in, then there is no point to have a login. So, in this lab you will understand how to handle access control for users and with jwt.

In this lab, you are required to apply all the obtained knowledge from the previous labs and create a complete applications.

This lab has grades 3,4 ,5 or U
Upload the code along with a report and fill in the following table and add it to your report with checkmarks in front of the tasks done.

Task	Grade 3	Grade 4	Grade 5
Create git repository for the lab	✓	✓	✓
Have incremental commits in repository	✓	✓	✓
Create a database with a table named “Users”	✓	✓	✓
Add 4 users to the table	✓	✓	✓
Create the path /admin	✓	✓	✓
Add Roles for users and admin	✓	✓	✓
Add roles for student 1, student 2, teacher		✓	✓
Create path /student1, /student2, /teacher		✓	✓
Render list of users in admin.ejs			✓
Register new users with bcrypt encryption			✓
Create dynamic route for user profiles			✓
Add readme.md file			✓

The purpose of this lab is the following:

- Understand Basic Access Control
- Understand how to verify jwt
- Understand middleware functions

2. Pre - requirements

- Understanding of rest api
- Have done lab 1,2 and 3.

Note some of the tasks use jwt, how to create a jwt cookie was explained in lab 3. Here we will just use them. If you need a reminder on what packages need to be installed and how to create an .env variable. Please go back to lab 3 and check again.

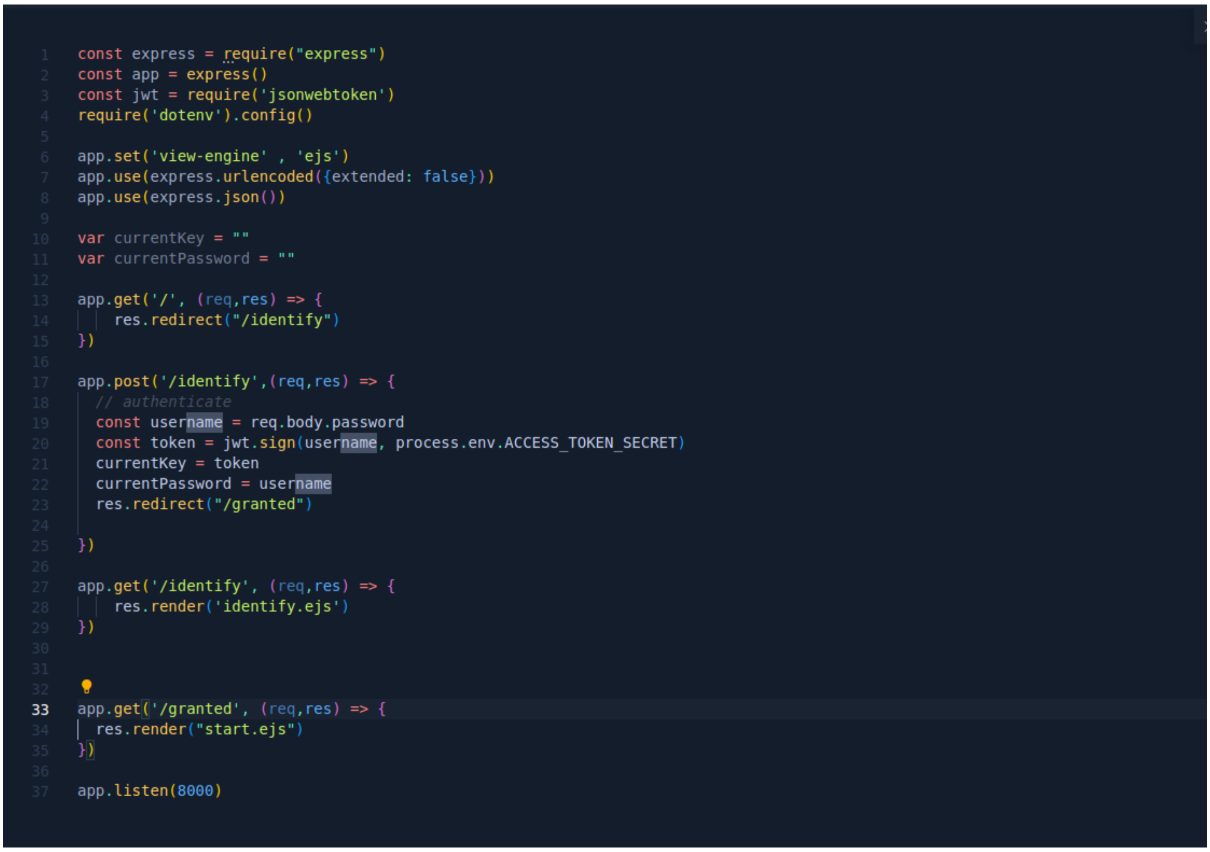
3. Instructions

3.1 Understanding Access control.

In this lab we will be learning how to use the JWT cookie we created in lab 3.

Before we start, you shall understand what Access Control means. Access control means that some users should be able to view some stuff and others should not. Think of your bank account, you should be the only one able to view your bank account.

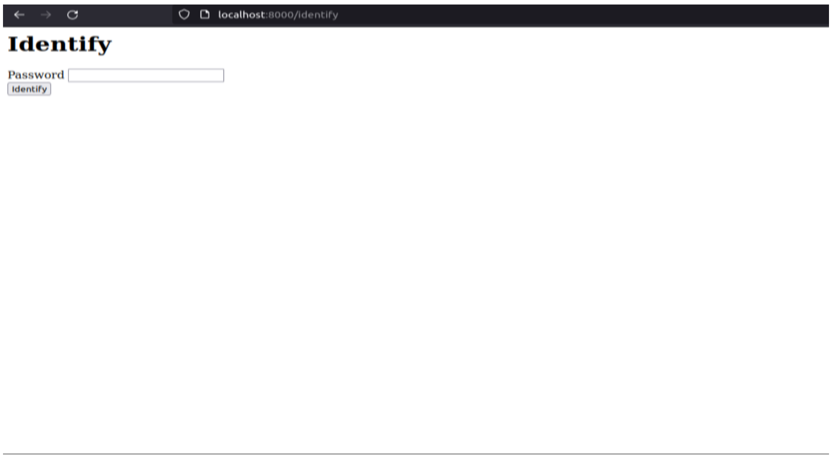
Now let us have this server code



```
const express = require("express")
const app = express()
const jwt = require('jsonwebtoken')
require('dotenv').config()

app.set('view-engine' , 'ejs')
app.use(express.urlencoded({extended: false}))
app.use(express.json())

var currentKey = ""
var currentPassword = ""

app.get('/', (req,res) => {
    res.redirect("/identify")
})

app.post('/identify',(req,res) => {
  // authenticate
  const username = req.body.password
  const token = jwt.sign(username, process.env.ACCESS_TOKEN_SECRET)
  currentKey = token
  currentPassword = username
  res.redirect("/granted")

})

app.get('/identify', (req,res) => {
    res.render('identify.ejs')
})


app.get('/granted', (req,res) => {
  res.render("start.ejs")
})

app.listen(8000)
```

Basically, we have a get route that redirects us directly to the /identify page. The point here is that a user has to identify before they can view the welcome page. Then we have a simple post route /identify where a user is being authenticated, you should know how to do this from the previous lab. When the user has been identified, they will then go to the /granted route. However, there is a small problem. You can access the granted page without being logged in because the granted page makes no checks for whether or not the user is logged in.

This is the identify page.

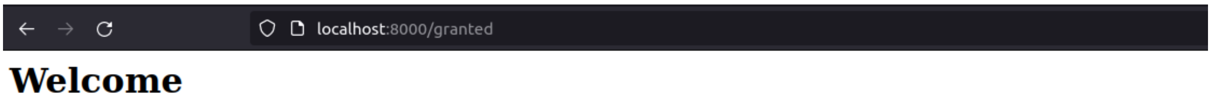


localhost:8000/identify

Identify

Password

If we change the url to localhost:8000/granted we can access the site without actually being identified. To solve this we can use middleware and jwt.

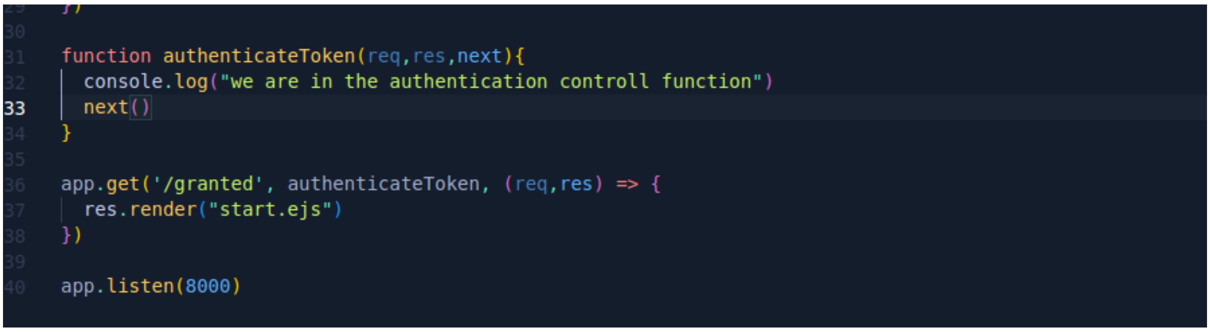


localhost:8000/granted

Welcome

3.2 Middleware and JWT

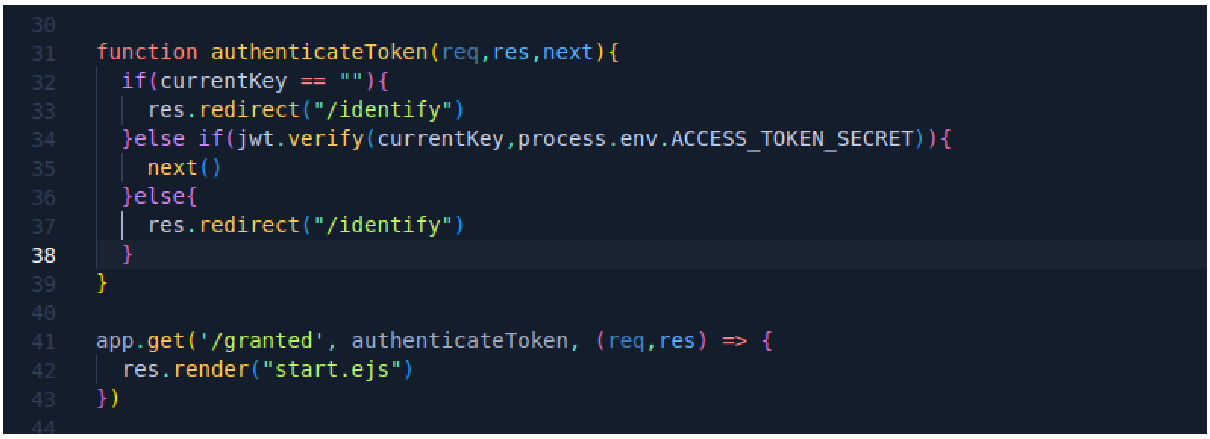
Middleware is the code in between the frontend requests and the backend rest routes. This can be used for a lot of tasks but in this case, we will be focusing on access control with jwt, groups and users. If we go back to our previous code to add middleware, we simply add a function in it.



```
})

function authenticateToken(req,res,next){
  console.log("we are in the authentication controll function")
  next()
}

app.get('/granted', authenticateToken, (req,res) => {
  res.render("start.ejs")
})

app.listen(8000)
```

As you can see, we added the function that simply says we are in authentication control. It then calls next; this means that it can continue to the next web page basically. To make this run before we go to the /granted page we have to add the function as a parameter in the route. If you run the program now and go to /granted you will see a message in the terminal.

Now we need to actually make the checks necessary to see that a user has been identified.



```
function authenticateToken(req,res,next){
  if(currentKey == ""){
    res.redirect("/identify")
  }else if(jwt.verify(currentKey,process.env.ACCESS_TOKEN_SECRET)){
    next()
  }else{
    res.redirect("/identify")
  }
}

app.get('/granted', authenticateToken, (req,res) => {
  res.render("start.ejs")
})
```

So now in authenticateToken function we have a few checks. First if the current key is “” then the user has not been identified. They get redirected to /identify. If jwt can verify the token we proceed to next and are shown granted. If the jwt can not verify it then they are again moved to /identify. Now run the code and try it yourself.

NOTE in code! jwt verify using the hash not the password. You can see in the code that we save the hash from the jwt sign. That is what we use together with access token in .env to make sure that the jwt is correct.

Important to note is that the same principle can be used to see if a user is a certain group in the database or if the current logged in user is allowed to view a page or not. This is what you will solve in this lab.

4. Tasks

Your task is to create a database with a couple of users and make sure that they have access to the correct files. Use the provided view files to solve the lab.

Tasks for Grade 3

1- Create git repository for the lab to publish you code to it.

2- The repository should have incremental commits.

3- Create a database with a table named “Users”:

- It should contain userID, role, name and password. The userID is a primary key, unique.
- Add 4 users to the table.
  - User1, userID: id1, name: user1, role: student and password: password
  - User2, userID: id2, name: user2, role: student and password: password2
  - User3, userID: id3, name: user3, role: teacher and password: password3
  - Admin, userID: admin, name: admin, role: admin and password: admin

4- Create the following path:

- /admin renders admin.ejs

5- Add the following Roles

- Users need to exist and be verified using jwt to view start.ejs
- Admin should be able to view admin.ejs

If a user tries to access something not allowed to access, it should be redirected to identify page and re-enter the current Password and current user.

If the user is authorized, continue with the request handling. If not, send a 401 Unauthorized response.

Tasks for Grade 4

You are required to do all previous tasks in addition to the following…

6- admin.ejs should render the list of users. Each user has userID , name , role and password.

7- Create the following paths:

- /student1 renders student1.ejs
- /student2 renders student2.ejs
- /teacher renders teacher.ejs

8- Add the following Roles

- Student1, teachers and admin should be able to view student1.ejs
- Student2, teachers and admin should be able to view student2.ejs
- Teacher and admin should be able to view teacher.ejs

If a user tries to access something not allowed to access, it should be redirected to identify page and re-enter the current Password and current user.

If the user is authorized, continue with the request handling. If not, send a 401 Unauthorized response.

Tasks for Grade 5

9- Create path for registration (the same way you did in the previous lab 3).

a. It renders register.ejs file
b. It should have the contents needed to register the user (userID, name, password, role)
    i. You can only choose roles as student or teacher (CANNOT register a new Admin)
c. The proper information should be saved in a database.
d. The password hash should be saved encrypted with bcrypt.
e. After the user has been registered you shall be redirected to /Identify.

10- Create a dynamic route that responds to requests with the user's ID

1- Create a route like "/users/:userId" for the user profile page.
2- When a user logs in to the website, they will be redirected to their own dynamic route.
3- The user profile page should not be viewable by other users.

11- Add a simple readme:md file in which you write down some instructions to run your website.

Markdown: Markdown is used all throughout the web for writing text that is displayed on websites, chat, forum, documentation and frequently used in git repos and on GitHub. Read briefly through the link below
https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax