Backend Development

Lab 3

1. Laboration goals

In this lab you will create a login and register for the backend.
Learning goals.

- Render pages and redirecting.
- Basic encryption
- Creating Tokens
- Using env variables
- Creating a login and register backend.

2. Pre - requirements

- Understanding REST API
- How to setup a database connection
- How to setup a node.js backend
- Using npm to install packages.

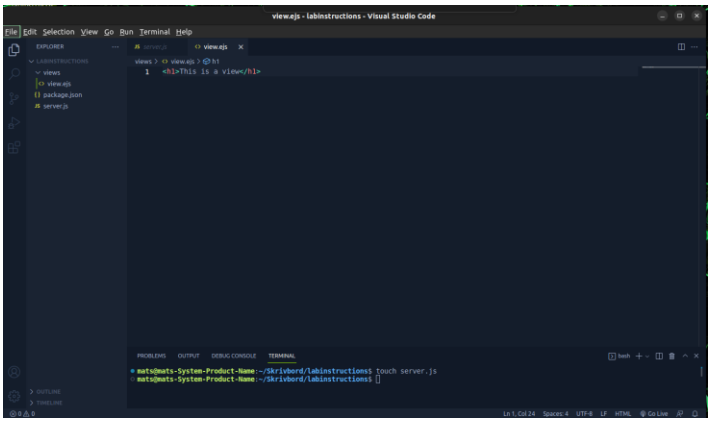
3. Instructions

3.1 Render pages

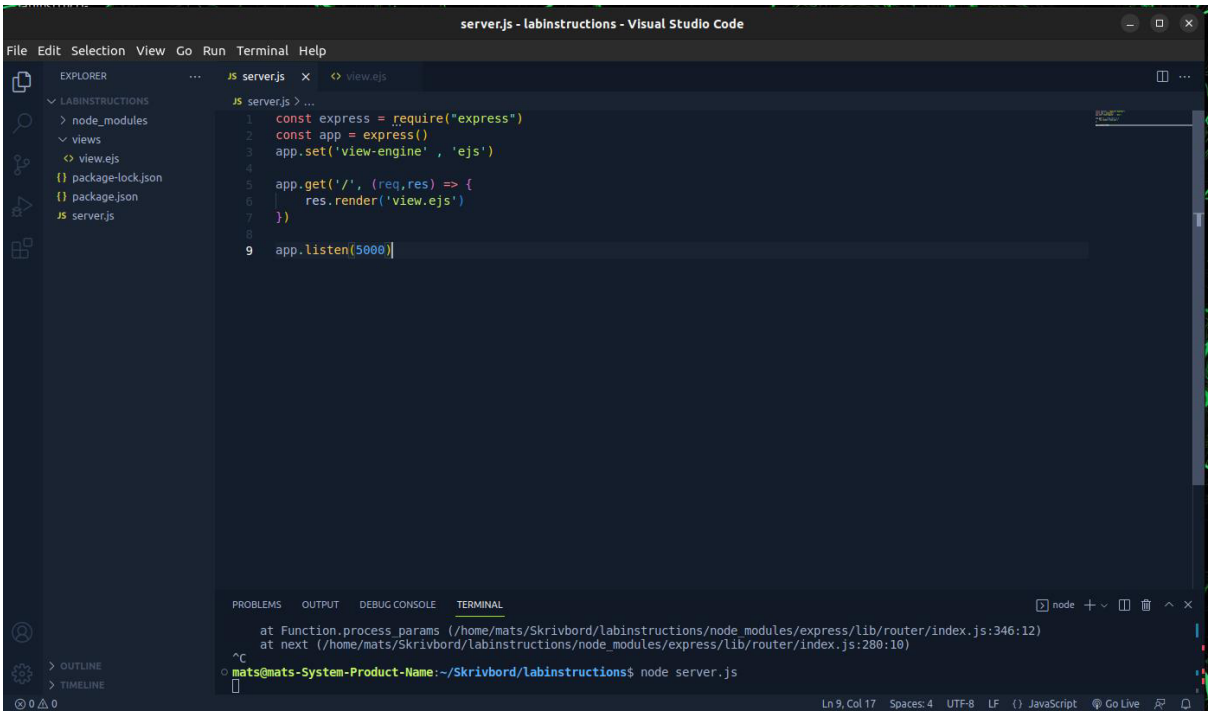
In rest routes we can redirect to pages. If for instance we where to press the help button, we can redirect to a specific view and if someone were to press another button we can redirect to another view.

To demonstrate this let us create a folder in visual studio code called views.
And in the view copy paste the following code into a file called view.ejs

```
<h1>This is a view</h1>
```



Now in the server.js file add the following snippets of code.

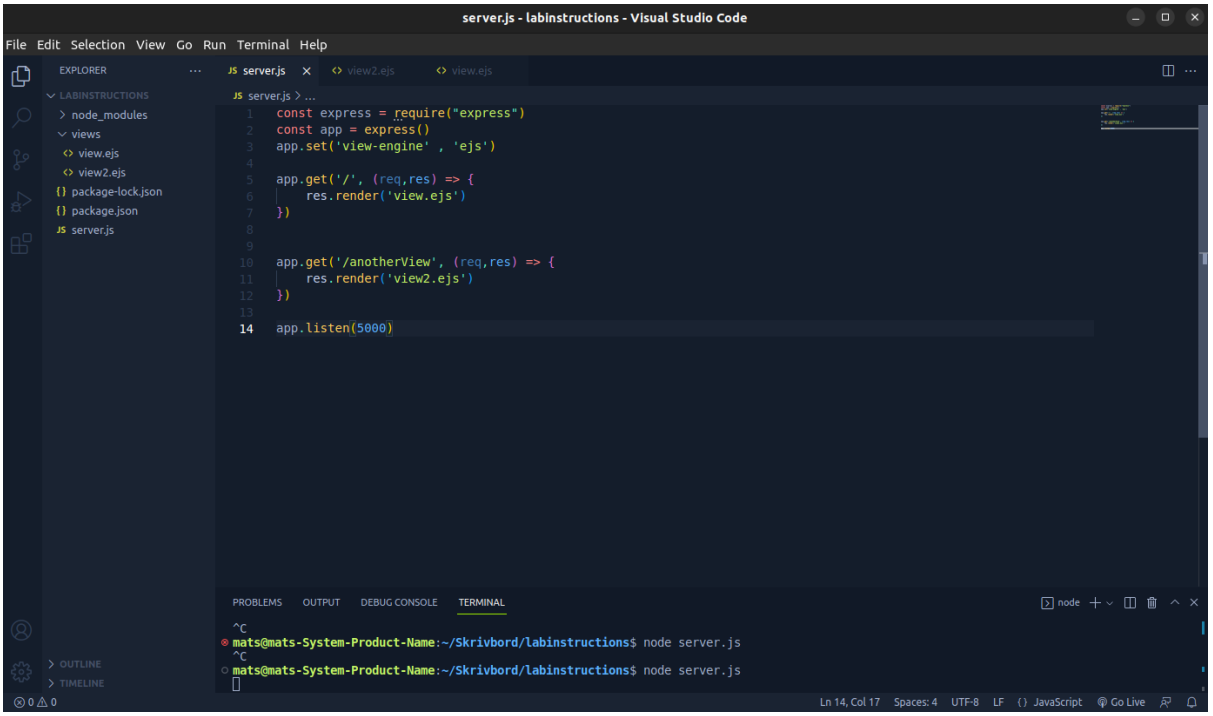


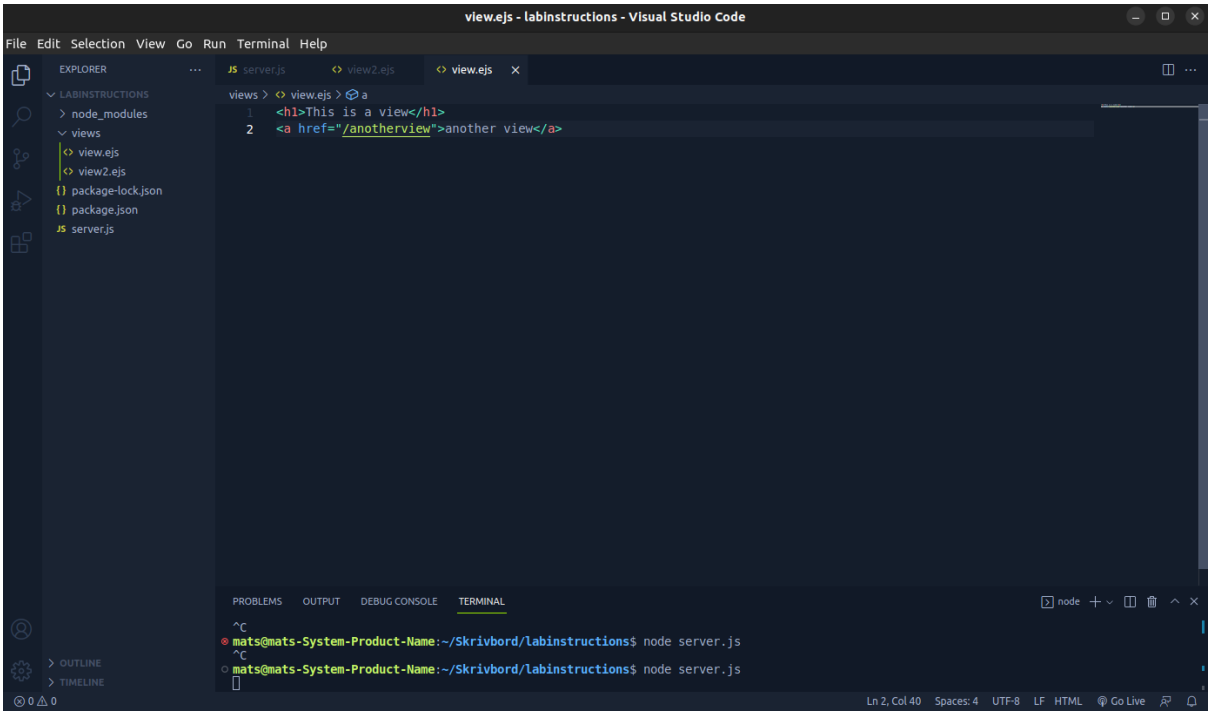
Here we just simply add express. Then we tell the express app to use view-engine and ejs. Afterwards make a simple rest route that will render the view we have created.

Run npm init
Install express, and ejs. Make sure you are in the right directory as always.

Run the program in the terminal by typing node server.js, go to localhost:5000 or whatever port you are running the program on. You should see the following.

Challenge task. In the current view folder create an href that takes you to a different view using a rest api. It should look something like this.





Try running the program and see what happens.

3.2 Encryption using bcrypt

Now we are going to learn about encryption. This is very important when it comes to user data in the backend. When you create a login it is best practice to save the password in encrypted format. So how do we encrypt something and compare it? Let us try it.

First run npm install bcrypt in the terminal.

when extended property is set to **false**, the URL-encoded data will instead be parsed with the query-string library.

Now we will create a different view.

```
Go  Run  Terminal  Help
server.js    view.ejs
views > view.ejs > form
<h1>Bcrypt</h1>
<form action="/" method="POST">
  <div>
    <label for="encyrpt">encrypt</label>
    <input type="text" id="encrypt" name="encrypt">
  </div>
  <div>
    <label for="compare">compare</label>
    <input type="compare" id="compare" name="compare">
  </div>
  <button type="submit">Send</button>
</form>
```

This view is basically a simple post form. It is used to post to a different post route.

In the server we will have to create two routes. One post route, where the post request will be made. One where we will render the page itself.

First we will make a simple if statement that will check if something to be encrypted is sent. Then it will encrypt it with bcrypt and save it to a local array. It looks something like this.

```
server.js ●   package.json   view.ejs
server.js > app.post('/') callback

app.use(express.urlencoded({extended: false}))
app.use(express.json())

var dbEncryption

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

app.post('/', async (req,res) => {

    console.log(req.body)
    if(req.body.encrypt != ''){
        try{
            dbEncryption = await bcrypt.hash(req.body.encrypt,10)
        }catch{
            console.log("error")

        }
    }

    req.method = 'GET'
    res.redirect("/")

})

app.listen(5000)
```

Now everytime we enter something to be encrypted it is encrypted with bcrypt and sent to the db string. Now let us make an if statement that encrypts something and checks if it is the same as in the string. Note we changed the req.method to get before redirecting to clear the data from the previous post. This is needed because we are redirecting. You can learn more about render and redirect here https://medium.com/@thorntonbrenden/to-render-or-to-redirect-that-is-the-question-b94e3bcac2e0.

```
server.js ×   package.json   view.ejs
server.js > app.post('/') callback

app.use(express.urlencoded({extended: false}))
app.use(express.json())

var dbEncryption

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

app.post('/', async (req,res) => {

    console.log(req.body)
    if(req.body.encrypt != ''){
        try{
            dbEncryption = await bcrypt.hash(req.body.encrypt,10)
        }catch{
            console.log("error")

        }
    }

    if(req.body.compare != ''){
        try{
            if(await bcrypt.compare(req.body.compare, dbEncryption)) {
                console.log("true")
              } else {
                console.log("false")
              }
        }catch{
            console.log("error")

        }
    }

    req.method = 'GET'
    res.redirect("/")

})

app.listen(5000)
```

Now run the program and do the following. Write encryption in the encrypt field and send. Then enter encrypt in the compare field. It should log true in the vs console.

```
server.js   package.json   view.ejs
server.js > app.post('/') callback

app.use(express.urlencoded({extended: false}))
app.use(express.json())

var dbEncryption

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

app.post('/', async (req,res) => {

    console.log(req.body)
    if(req.body.encrypt != ''){
        try{
            dbEncryption = await bcrypt.hash(req.body.encrypt,10)
        }catch{
            console.log("error")

        }
    }

    if(req.body.compare != ''){
        try{
            if(await bcrypt.compare(req.body.compare, dbEncryption)) {
                console.log("true")
              } else {
                console.log("false")
              }
        }catch{
            console.log("error")

        }
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

mats@mats-System-Product-Name:~/Skrivbord/labinstructions$ node server.js
[Object: null prototype] { encrypt: 'encrypt', compare: '' }
[Object: null prototype] { encrypt: '', compare: 'encrypt' }
true
```

3.2 jsonwebtokens and enviorment variables.

Now we will introduce environment variables and jsonwebtokens. This is important for security. We will talk more about it in the next lab. Here you will simply learn how to create one. First we need to install jsonwebtoken and dotenv using npm. Let us start with environment variables since they are needed when we use the json web tokens. First create a .env file in the visual studio code project.

```
EXPLORER
LABINSTRUCTIONS
  node_modules
  views
  .env
  package-lock.json
  package.json
  server.js
```

Enter this into the .env file

```
TEST = 'hej this is a test enviorment variable'
```

Now in our rest route / we will simply log this string.

```
const express = require("express")
const app = express()
app.set('view-engine' , 'ejs')
const bcrypt = require('bcrypt')
const jwt = require("jsonwebtoken")
require("dotenv").config()

app.use(express.urlencoded({extended: false}))
app.use(express.json())


app.get('/', (req,res) => {
    res.render("view.ejs")
    console.log(process.env.TEST)
})


app.listen(5000)
```

Runt it using node server.js and visit the website. The string should be logged in the terminal.

Now you understand environment variables. Let us use one to create a json web token. Suggested reading to understand json web tokens

https://www.logicmonitor.com/blog/what-are-json-web-tokens

To generate our secret token we can run crypto. To do this go to the vs code terminal and enter node to get to interactive mode. then enter require('crypto').randomBytes(64).toString('hex'). Press control+c twice to exit interactive mode.This generates 64 random bytes in hex. It should look like this.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

mats@mats-System-Product-Name:~/Skrivbord/labinstructions$ node
Welcome to Node.js v12.22.9.
Type ".help" for more information.
> require('crypto').randomBytes(64).toString('hex')
'5bab6f48e8343cdb0f8d9f92dd3635b05048809a65f8ff86ea9e38e7ee02d5a524db01a6702b1944f5c84fe12a0c3857a8d85ad64c33fabd13027523267aa50b'
>
```

Add this to your environment.

```
n  Terminal  Help
JS server.js    .env    {} package.json    <> view.ejs
.env
1   TOKEN = '5bab6f48e8343cdb0f8d9f92dd3635b05048809a65f8ff86ea9e38e7ee02d5a524db01a6702b1944f5c84fe12a0c3857a8d85ad64c33fabd13027523267aa50b'
```

Now we can simply create a sign with jtw in our rest api. This creates our token and the code looks like this.

```
n  Terminal  Help
JS server.js    .env    {} package.json    <> view.ejs
JS server.js > ...
1   const express = require("express")
2   const app = express()
3   app.set('view-engine' , 'ejs')
4   const bcrypt = require('bcrypt')
5   const jwt = require("jsonwebtoken")
6   require("dotenv").config()
7
8   app.use(express.urlencoded({extended: false}))
9   app.use(express.json())
10
11
12  app.get('/', (req,res) => {
13      res.render("view.ejs")
14      var token = jwt.sign("username",process.env.TOKEN)
15      console.log(token)
16  })
17
18
19  app.listen(5000)
```

Now run it and you will see that you get a token.

4. Task

Using what you have learned in the instructions. You are given 4 view models. You should create a login backend. Where the user has to create an account and then log in. Remember to use the views given to you. If you want to create your own to make it more appealing you are welcome to do so.

5. Requirements

There should be 4 rest routes.

Database

- user table
- table should contain username and password
- password should be encrypted when saved with bcrypt.

app.get("/")

- Here you should just redirect to "/LOGIN".

app.post"/LOGIN"

- Here you should make a if else check to see if the input is correct. The username from the front end will be in req.body.name and password in req.body.password
- If the user does not exist you should render fail.ejs
- If the user does exist you should render start.ejs
- The password should be compared using bcrypt
- Finally, if the user is logged in you should create a jwt token that shall be saved in a string and logged.

app.get("/LOGIN")

- Here you shall simply render login.ejs

app.post("/REGISTER")

- Here you shall register the user.
- It shall be saved in a database, a proper one. The password shall be saved encrypted with bcrypt.
- After the user has been registered you shall be redirected to /LOGIN.

app.get("/REGISTER")

- render register.ejs file