Name: Hajra Zafar

DHC-Id: 490

# WEEK 2:

| |
|---|
| ▪ validator.js (for input validation) |
| ▪ bcrypt (for password hashing) |
| ▪ jsonwebtoken (for authentication) |
| ▪ helmet (for secure headers) |

# IMPLEMENTING SECURITY MEASURES

## 1. Fix Vulnerabilities

### ● Sanitize and Validate Inputs:

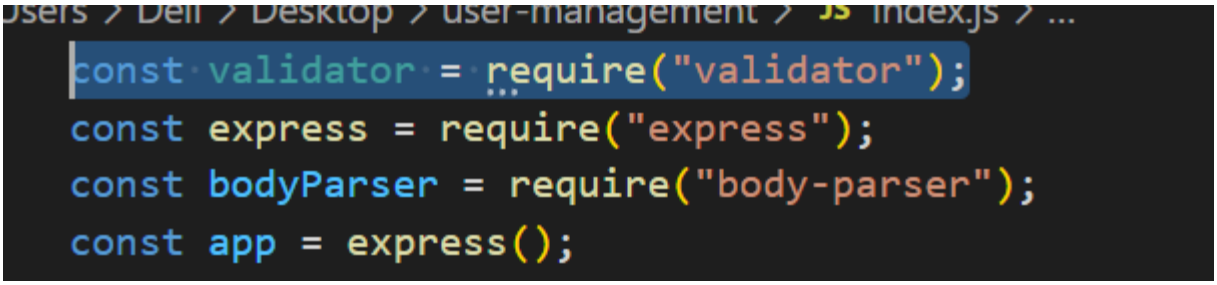
**Install the first security package (validator)**

We have to open up the the user management app in terminal and run this command to install the validators

```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Dell\Desktop\user-management> npm install validator

added 1 package, and audited 77 packages in 8s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
npm notice
npm notice New major version of npm available! 10.8.2 -> 11.5.2
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.5.2
npm notice To update run: npm install -g npm@11.5.2
npm notice
PS C:\Users\Dell\Desktop\user-management>
```

# Next Step: Use validator in your code

Add this on header of the index.js:




# Update your signup route to validate input:

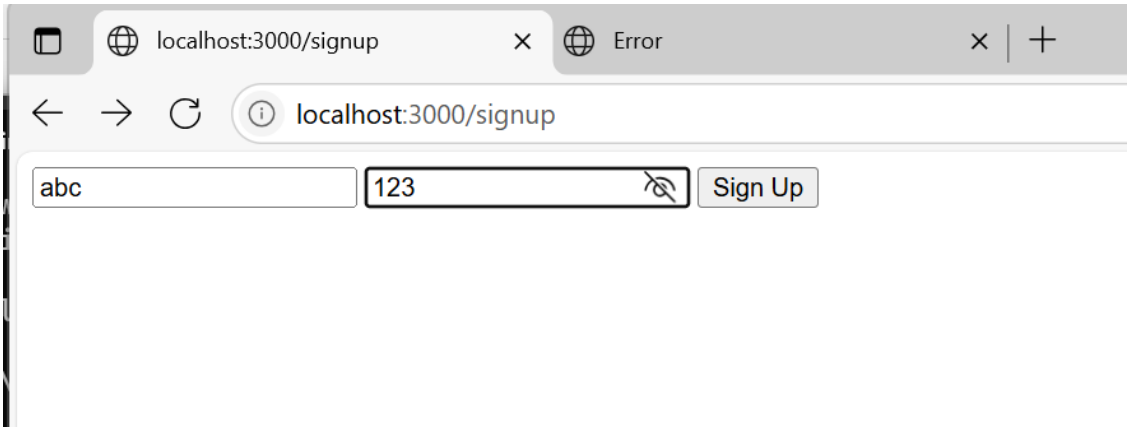
Then replace this part:

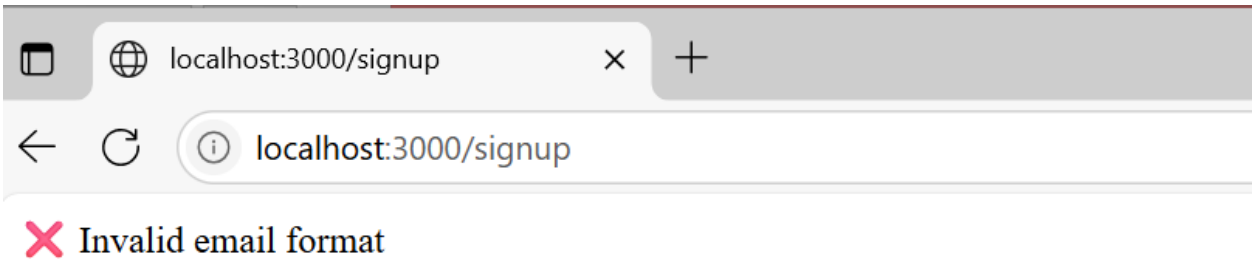
By this:

```
  res.render("signup");
});

app.post("/signup", (req, res) => {
  const { username, password } = req.body;

  // Validate email format
  if (!validator.isEmail(username)) {
    return res.status(400).send("❌ Invalid email format");
  }

  // Validate strong password
  if (!validator.isStrongPassword(password)) {
    return res.status(400).send("❌ Weak password. Use mix of letters, numbers & symbols");
  }

  users.push({ username, password });
  res.send("✅ User registered: " + username);
});

// Login page
app.get("/login", (req, res) => {
```

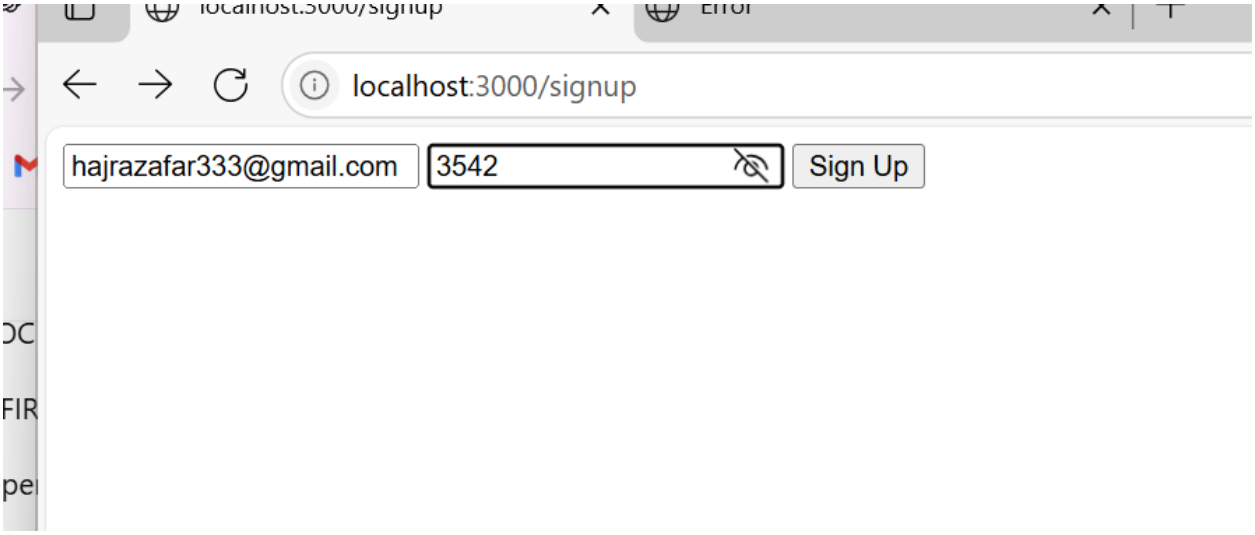
# verification :

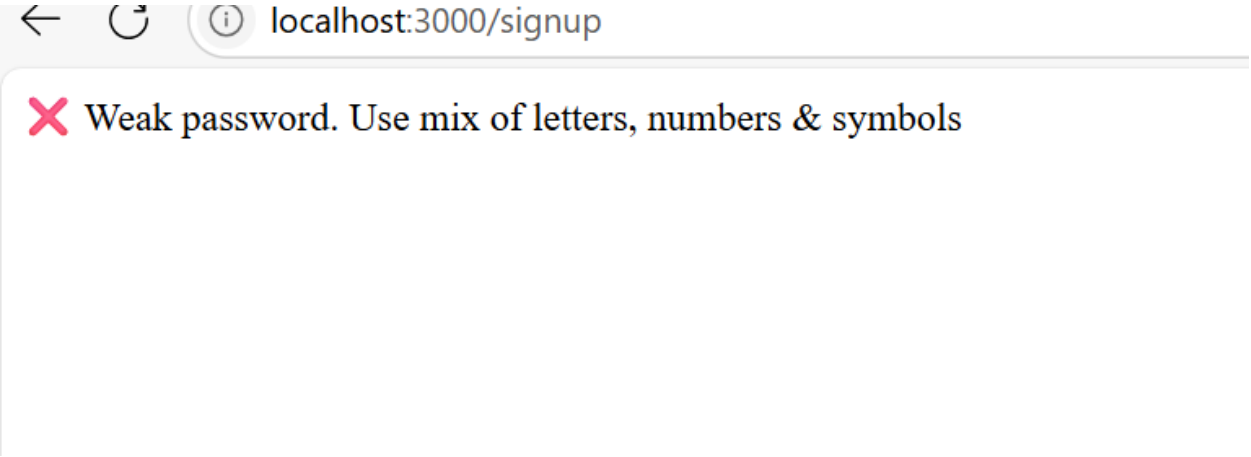
Tried this





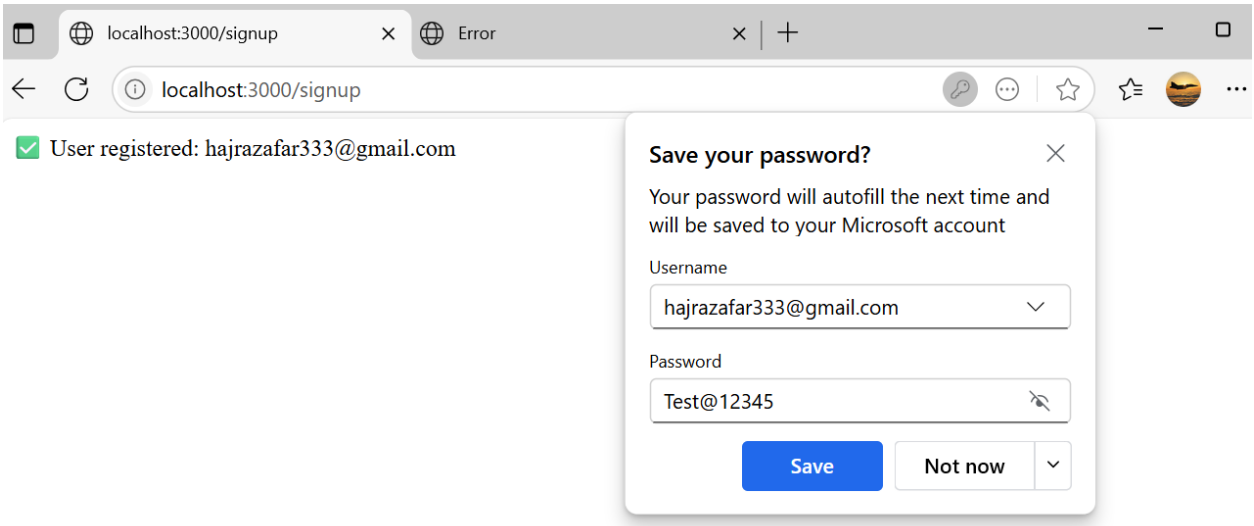

**Another example :**






## 3rd and correct case:

If we use correct mail and password including uppercase ,special characters and numeric we'll get the correct result:




# • Password Hashing:

Right now our app still saves passwords in **plain text** (bad practice). Let's fix that.First We have to open up the the user management app in terminal and run this command to install the bcrypt library.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Dell\Desktop\user-management> npm install bcrypt

added 3 packages, and audited 80 packages in 9s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\Dell\Desktop\user-management>
```

# Next step Import bcrypt in index.js:


# Update Signup Route:

Change from this :

```
  }

  // Validate strong password
  if (!validator.isStrongPassword(password)) {
    return res.status(400).send("❌ Weak password. Use mix of letters,
  }

  users.push({ username, password });
  res.send("✅ User registered: " + username);
});

// Login page
app.get("/login", (req, res) => {
  res.render("login");
});
```

To this:

```
  if (!validator.isStrongPassword(password)) {
    return res.status(400).send("❌ Weak password. Use mix of letters, numbers & symbols");
  }

  // Hash password before saving
bcrypt.hash(password, 10, (err, hashedPassword) => {
  if (err) return res.status(500).send("Error hashing password");

  users.push({ username, password: hashedPassword });
  res.send("✅ User registered securely: " + username);
});

});

// Login page
app.get("/login", (req, res) => {
```

# Update Login Route

Currently login checks plain passwords. Change it to check **hashed password**:

Lets Find our login route and replace

From this:

```
// Login page
app.get("/login", (req, res) => {
  res.render("login");
});

app.post("/login", (req, res) => {
  const user = users.find(
    u => u.username === req.body.username && u.password === req.body.password
  );
  if (user) {
    res.send("Welcome " + user.username);
  } else {
    res.send("Invalid credentials");
  }
});

app.listen(3000, () => console.log("App running at http://localhost:3000"));
```

To this :

```
40  });
41
42  app.post("/login", (req, res) => {
43    const { username, password } = req.body;
44    const user = users.find(u => u.username === username);
45
46    if (!user) {
47      return res.send("❌ Invalid credentials");
48    }
49
50    // Compare entered password with hashed password
51    bcrypt.compare(password, user.password, (err, result) => {
52      if (err) return res.status(500).send("Error during login");
53
54      if (result) {
55        res.send("✅ Welcome " + user.username);
56      } else {
57        res.send("❌ Invalid credentials");
58      }
59    });
60  });
```

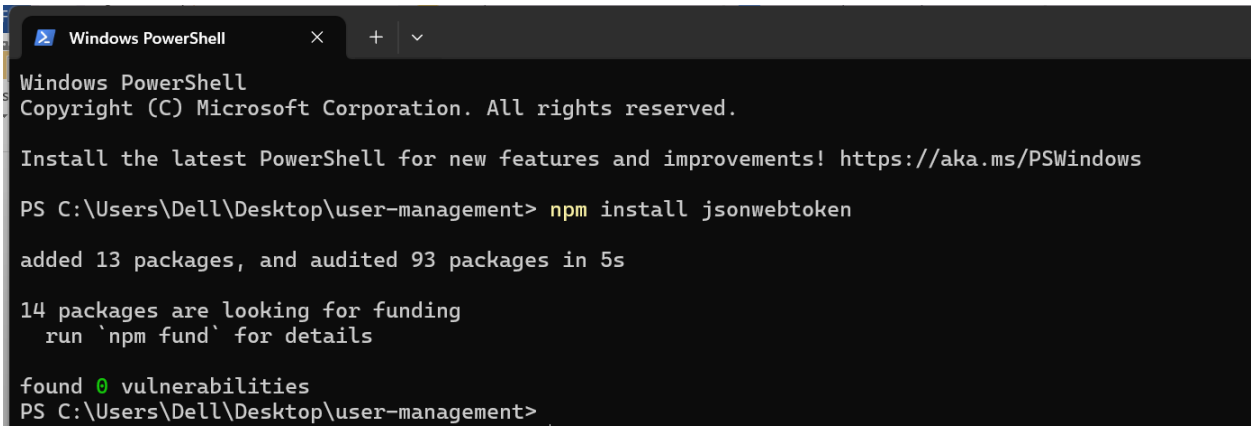
**Password Hashing:**

Implemented bcrypt.hash() to store hashed passwords instead of plaintext. Passwords are now stored as secure hashes (e.g., $2b$10$...) inside the users array.

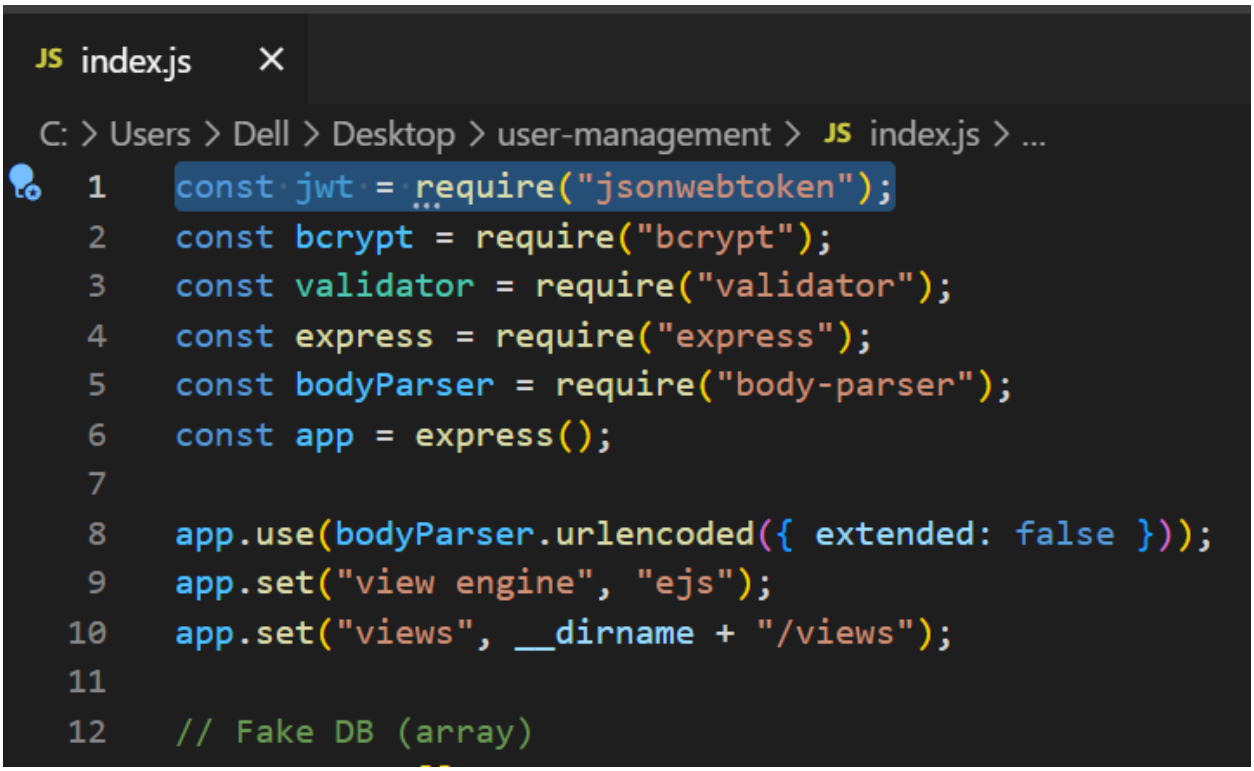
# 2. Enhance Authentication

Now we'll add **JSON Web Tokens (JWT).**

# Install JWT:




# Import JWT in index.js:



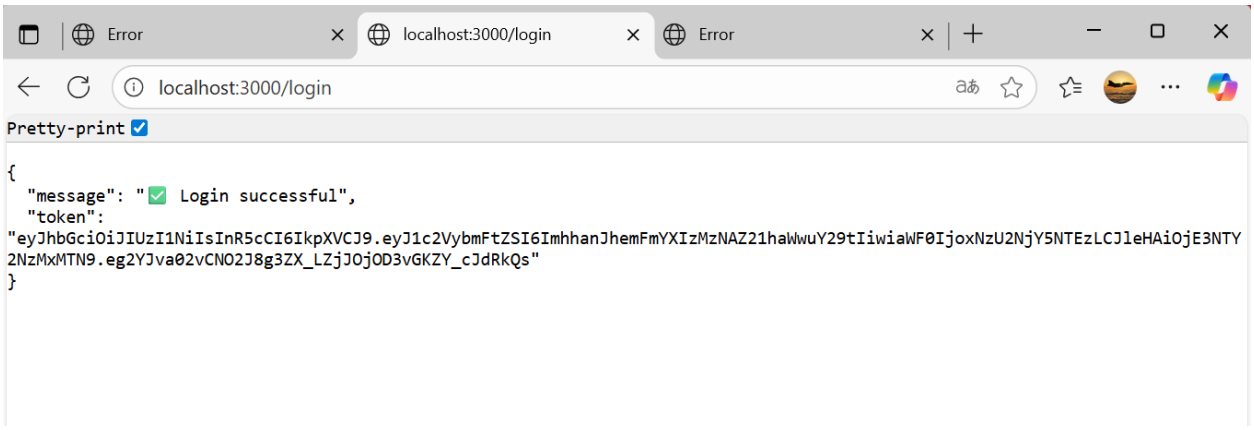
# Update Login Route:

```
});

app.post("/login", (req, res) => {
  const { username, password } = req.body;
  const user = users.find(u => u.username === username);

  if (!user) {
    return res.send("❌ Invalid credentials");
  }

  // Compare entered password with hashed password
  bcrypt.compare(password, user.password, (err, result) => {
    if (err) return res.status(500).send("Error during login");

    if (result) {
      res.send("✅ Welcome " + user.username);
    } else {
      res.send("❌ Invalid credentials");
    }
  });
});
app.listen(3000, () => console.log("App running at http://localhost:3000"));

```

# By:

```
});

app.post("/login", (req, res) => {
  const { username, password } = req.body;
  const user = users.find(u => u.username === username);

  if (!user) {
    return res.send("❌ Invalid credentials");
  }

  // Compare entered password with hashed password
  bcrypt.compare(password, user.password, (err, result) => {
    if (err) return res.status(500).send("Error during login");

    if (result) {
      // Generate JWT token
      const token = jwt.sign(
        { username: user.username },
        "super-secret-key",   // ⚠️ in real apps, use env variable
        { expiresIn: "1h" }
      );

      res.json({ message: "✅ Login successful", token: token });
    } else {
      res.send("❌ Invalid credentials");
    }
  });
});
```

# Verification:

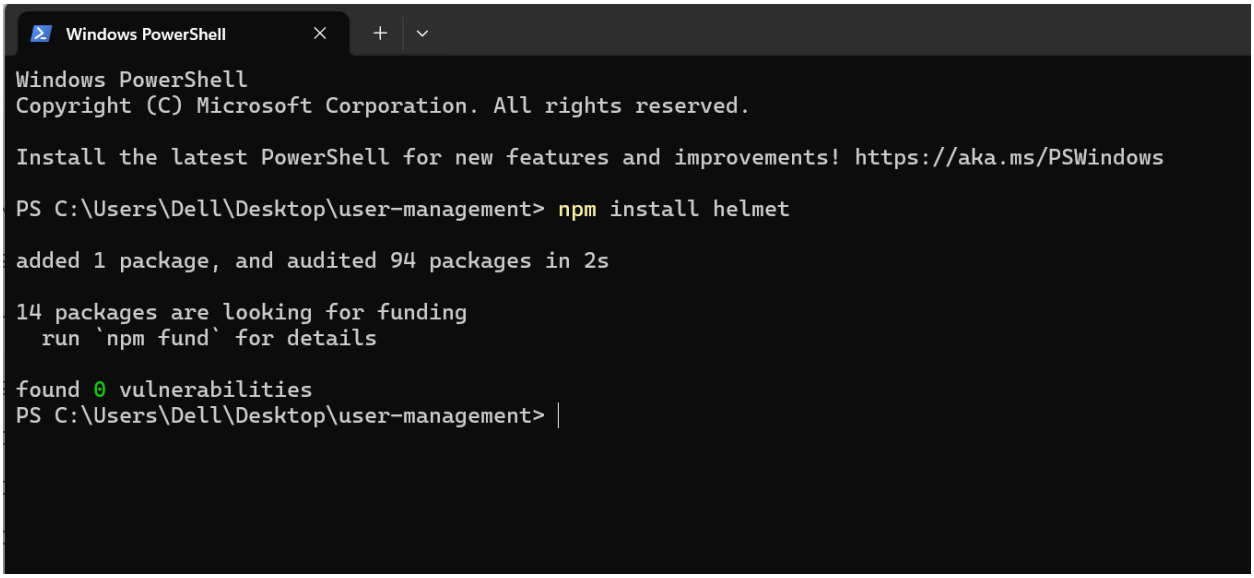



That means:

- **bcrypt** worked (password checked securely)
- **JWT** worked (token generated successfully)

# 3. Secure Data Transmission

# Install helmet:




# Import and enable it in index.js:

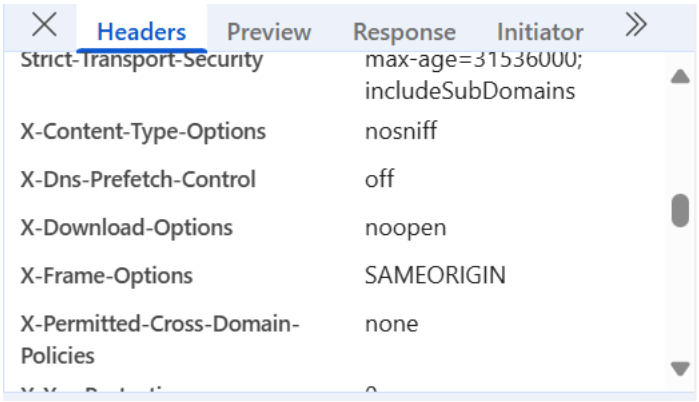
Users > Dell > Desktop > user-management > JS index.js > ...

```
const helmet = require("helmet");
const jwt = require("jsonwebtoken");
const bcrypt = require("bcrypt");
const validator = require("validator");
const express = require("express");
const bodyParser = require("body-parser");
const app = express();

app.use(bodyParser.urlencoded({ extended: false }));
app.set("view engine", "ejs");
app.set("views", __dirname + "/views");

app.use(helmet());

// Fake DB (array)
```

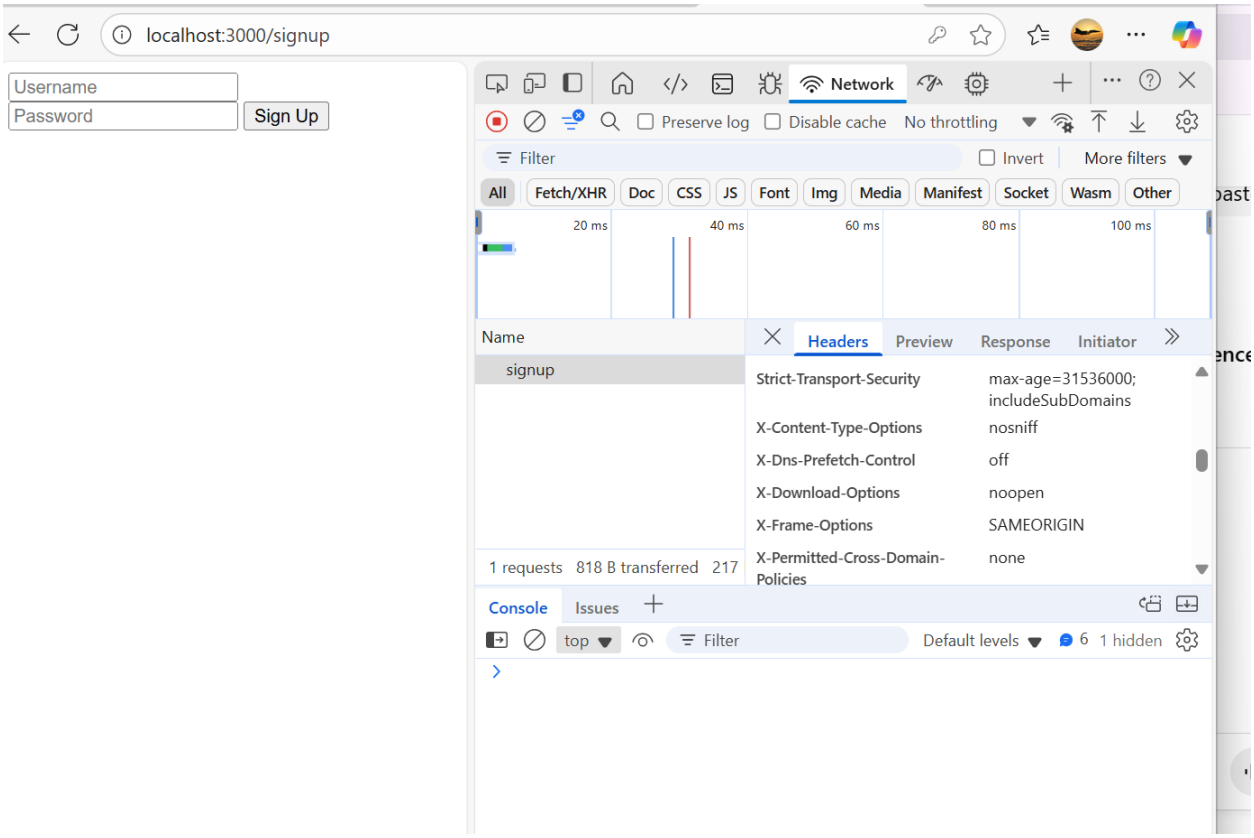
# Verification:

Open browser → press f12→ ctrl R→below “name”→ click signup→headers→response header

# Zoom in:




# Zoom out:




# Summary:

In Week 2, I implemented input validation, password hashing, JWT-based authentication, and secured HTTP headers with Helmet. These fixes addressed the vulnerabilities found in Week 1 and significantly improved the application's security posture.