Implementing Security Measures Report

This report is the follow-up report of the Security Assessment task, in which vulnerabilities are identified and security measures are presented. In this task, in the second week of the internship, a few of those security measures are practically implemented.

Developers Hub Cybersecurity Internship Task June 2025 Week ***2***

Internee name: ***Athar Imran***

https://github.com/atharimran728/Web-Application-Security-Strengthening/tree/main

---

GOAL: **Fix the identified vulnerabilities**

There will be four steps, fixing some vulnerabilities identified, as follows:

- ☑ ~~Sanitize and Validate Inputs~~
- ☑ ~~Hash Passwords with bcrypt~~
- ☑ ~~Implement JWT Authentication~~
- ☑ ~~Secure HTTP Headers~~

Note

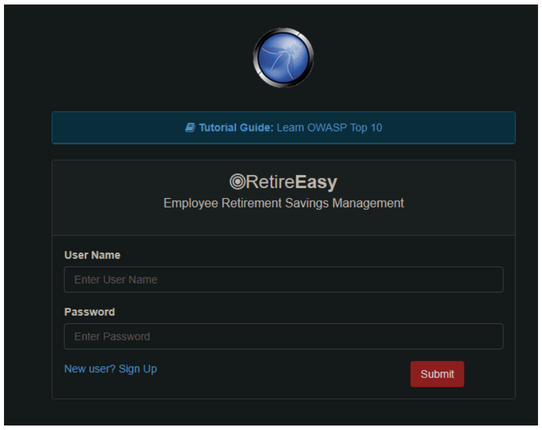
Because we are fixing the vulnerability, the web application we previously worked on only focuses on the exploitation of vulnerabilities. So, here to complete this week's task, we are employing a different web application, which is also designed to fix vulnerabilities - OWASP NodeGoat.

❖ Setting up a Web Application:

(Alternatively, follow the official tutorial: https://github.com/OWASP/NodeGoat)

1. Download and install Docker from the official source. After finishing the installation, ensure that it was installed correctly
2. Now clone NodeGoat: `git clone https://github.com/OWASP/NodeGoat.git`

3. Got the NodeGoat directory and built Docker image using: `docker-compose build`. This command reads the `Dockerfile` and `docker-compose.yml` to build the necessary images for the application and the database.
4. Run the application using `docker-compose up`. And access at http://localhost:4000/. Now our application starts listening on http port 4000:

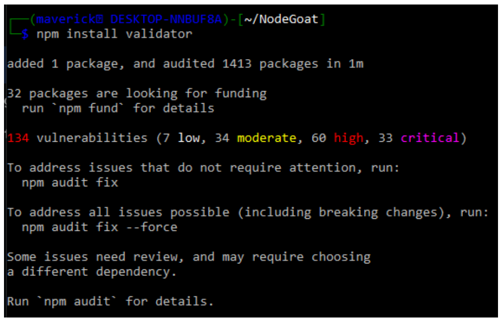


Now we will start focusing on our main tasks.

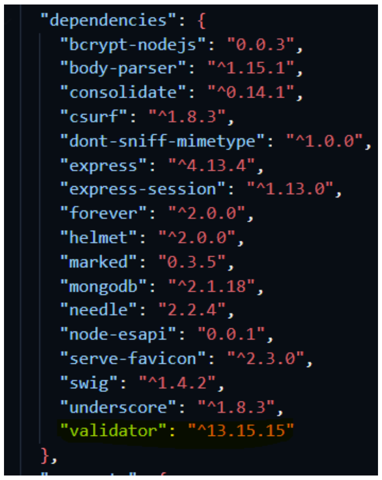
1- Sanitizing and Validating Inputs:

A. **Install `validator`:**

   a. Run the command `npm install validator` in the main NodeGoat directory.



```
┌──(maverick㉿ DESKTOP-NNBUF8A)-[~/NodeGoat]
└─$ npm install validator

added 1 package, and audited 1413 packages in 1m

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

134 vulnerabilities (7 low, 34 moderate, 60 high, 33 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues possible (including breaking changes), run:
  npm audit fix --force

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
```



```
"dependencies": {
  "bcrypt-nodejs": "0.0.3",
  "body-parser": "^1.15.1",
  "consolidate": "^0.14.1",
  "csurf": "^1.8.3",
  "dont-sniff-mimetype": "^1.0.0",
  "express": "^4.13.4",
  "express-session": "^1.13.0",
  "forever": "^2.0.0",
  "helmet": "^2.0.0",
  "marked": "0.3.5",
  "mongodb": "^2.1.18",
  "needle": "2.2.4",
  "node-esapi": "0.0.1",
  "serve-favicon": "^2.3.0",
  "swig": "^1.4.2",
  "underscore": "^1.8.3",
  "validator": "^13.15.15"
},
```

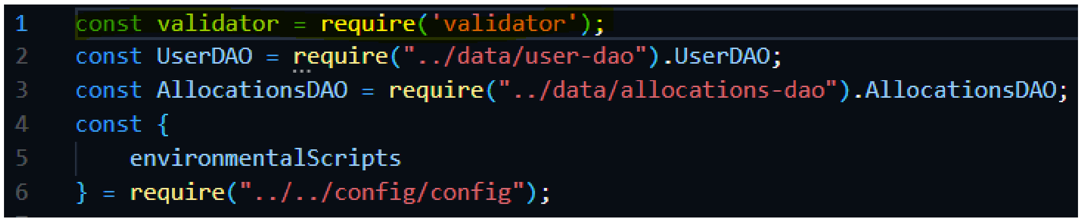
   b. Update and rebuild Docker image using (because we are on WSL): `docker-compose build --no-cache`

```
docker-compose up
```

B. **Update the code:**

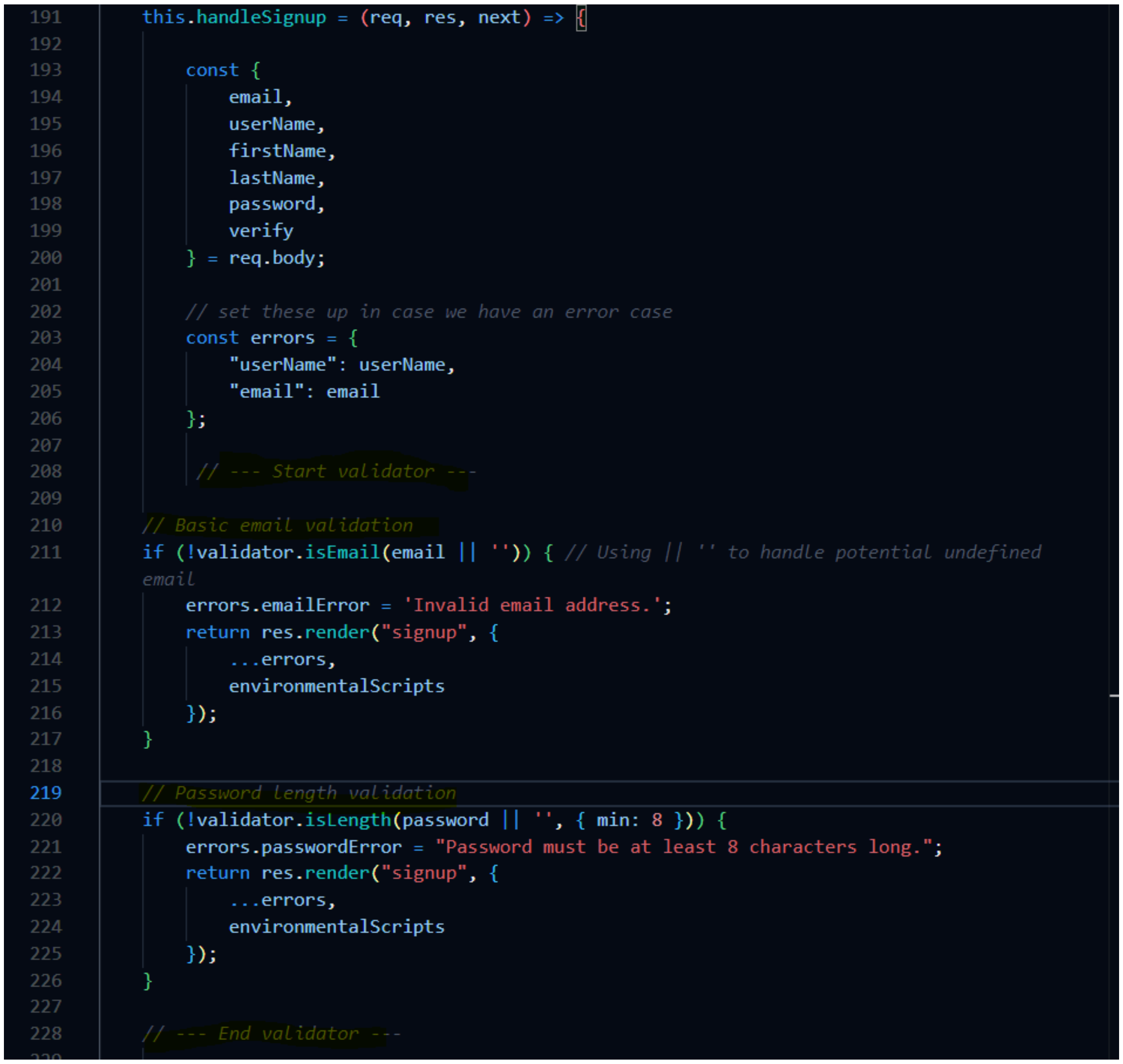
a. Find the .js file that contains the code of signup. (Search for `POST signup-related` codes). In our case it's session.js.

b. Add validator dependency at the start of the code:



```javascript
const validator = require('validator');
const UserDAO = require("../data/user-dao").UserDAO;
const AllocationsDAO = require("../data/allocations-dao").AllocationsDAO;
const {
    environmentalScripts
} = require("../../config/config");
```

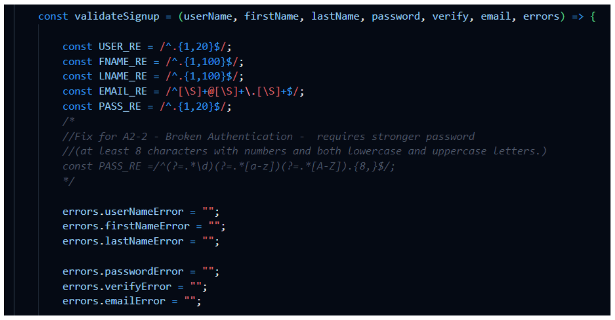
c. Find the function that handles Signup, in this case: `handleSignup,` and add the code lines to validate email and password:



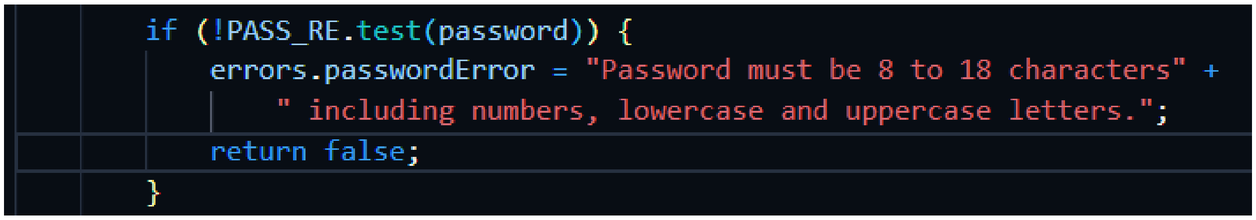
```javascript
    this.handleSignup = (req, res, next) => {

        const {
            email,
            userName,
            firstName,
            lastName,
            password,
            verify
        } = req.body;

        // set these up in case we have an error case
        const errors = {
            "userName": userName,
            "email": email
        };

         // --- Start validator ---

    // Basic email validation
    if (!validator.isEmail(email || '')) { // Using || '' to handle potential undefined email
        errors.emailError = 'Invalid email address.';
        return res.render("signup", {
            ...errors,
            environmentalScripts
        });
    }

    // Password length validation
    if (!validator.isLength(password || '', { min: 8 })) {
        errors.passwordError = "Password must be at least 8 characters long.";
        return res.render("signup", {
            ...errors,
            environmentalScripts
        });
    }

    // --- End validator ---
```

d. Alternatively, after understanding our target machine code, I will craft this code (if not present) to make the server not accept passwords outside of a secure bracket:

```
if (!PASS_RE.test(password)) {
            errors.passwordError = "Password must be 8 to 18 characters" +
                " including numbers, lowercase and uppercase letters.";
            return false;
```



```
const validateSignup = (userName, firstName, lastName, password, verify, email, errors) => {

    const USER_RE = /^.{1,20}$/;
    const FNAME_RE = /^.{1,100}$/;
    const LNAME_RE = /^.{1,100}$/;
    const EMAIL_RE = /^[\S]+@[\S]+\.[\S]+$/;
    const PASS_RE = /^.{1,20}$/;
    /*
    //Fix for A2-2 - Broken Authentication -  requires stronger password
    //(at least 8 characters with numbers and both lowercase and uppercase letters.)
    const PASS_RE =/^(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}$/;
    */

    errors.userNameError = "";
    errors.firstNameError = "";
    errors.lastNameError = "";

    errors.passwordError = "";
    errors.verifyError = "";
    errors.emailError = "";
```



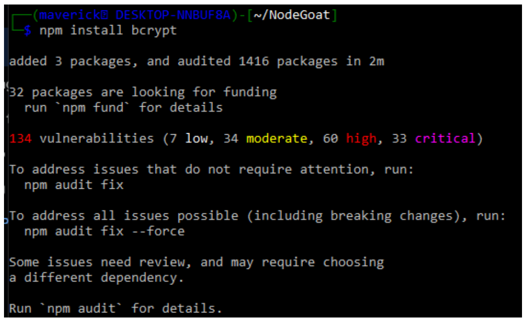
```
        if (!PASS_RE.test(password)) {
            errors.passwordError = "Password must be 8 to 18 characters" +
                " including numbers, lowercase and uppercase letters.";
            return false;
        }
```

2- Hashing Passwords with bcrypt:

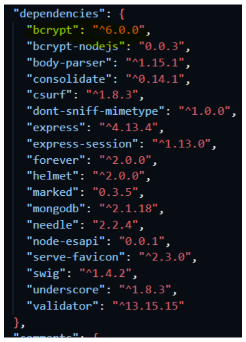
In this section, we will add the code in our signup and login .js files of NodGoat that will encrypt the password to store it into the DataBase.

C. Install `bcrypt`:

a. Use the command `npm install bcrypt` in the main NodeGoat directory.



```
┌──(maverick㉿ DESKTOP-NNBUF8A)-[~/NodeGoat]
└─$ npm install bcrypt

added 3 packages, and audited 1416 packages in 2m

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

134 vulnerabilities (7 low, 34 moderate, 60 high, 33 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues possible (including breaking changes), run:
  npm audit fix --force

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
```

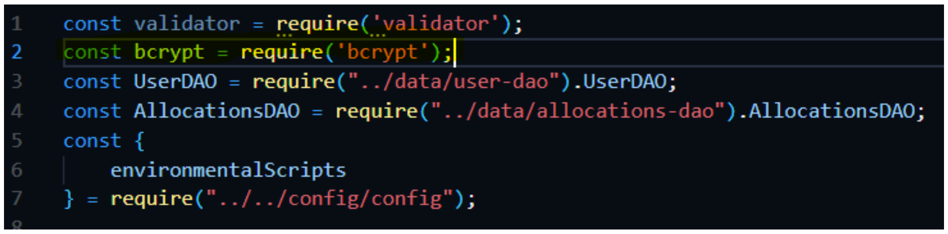


```
"dependencies": {
  "bcrypt": "^6.0.0",
  "bcrypt-nodejs": "0.0.3",
  "body-parser": "^1.15.1",
  "consolidate": "^0.14.1",
  "csurf": "^1.8.3",
  "dont-sniff-mimetype": "^1.0.0",
  "express": "^4.13.4",
  "express-session": "^1.13.0",
  "forever": "^2.0.0",
  "helmet": "^2.0.0",
  "marked": "0.3.5",
  "mongodb": "^2.1.18",
  "needle": "2.2.4",
  "node-esapi": "0.0.1",
  "serve-favicon": "^2.3.0",
  "swig": "^1.4.2",
  "underscore": "^1.8.3",
  "validator": "^13.15.15"
},
```

b. Again, update and rebuild the Docker image:
```
docker-compose build --no-cache
docker-compose up
```

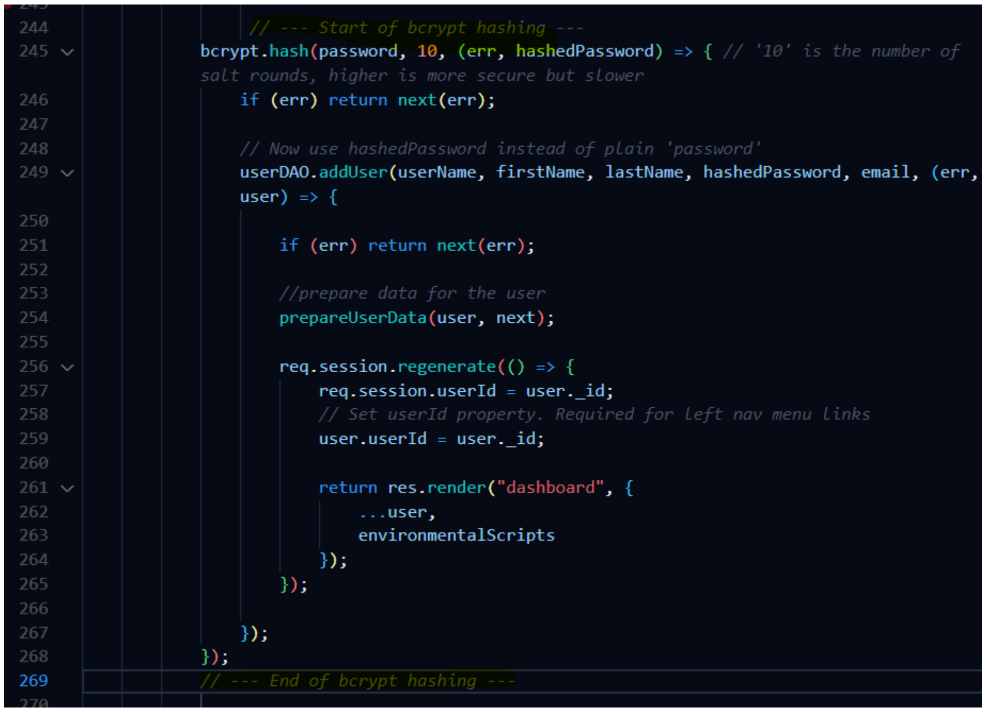
D. **Update the code:**

a. Again in session.js, first add those lines at the start of the code:



```
const validator = require('validator');
const bcrypt = require('bcrypt');
const UserDAO = require("../data/user-dao").UserDAO;
const AllocationsDAO = require("../data/allocations-dao").AllocationsDAO;
const {
    environmentalScripts
} = require("../../config/config");
```

b. Add the lines of code to encrypt the credentials on the signup step:

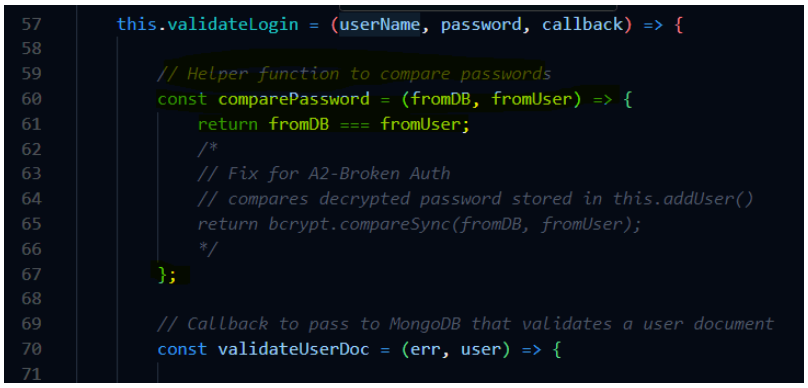


```
            // --- Start of bcrypt hashing ---
        bcrypt.hash(password, 10, (err, hashedPassword) => { // '10' is the number of salt rounds, higher is more secure but slower
            if (err) return next(err);

            // Now use hashedPassword instead of plain 'password'
            userDAO.addUser(userName, firstName, lastName, hashedPassword, email, (err, user) => {

                if (err) return next(err);

                //prepare data for the user
                prepareUserData(user, next);

                req.session.regenerate(() => {
                    req.session.userId = user._id;
                    // Set userId property. Required for left nav menu links
                    user.userId = user._id;

                    return res.render("dashboard", {
                        ...user,
                        environmentalScripts
                    });
                });

            });
        });
        // --- End of bcrypt hashing ---
```

These code lines are written under the same `handlesignup` function.

c. To encrypt credentials at the login step, find the `handleLoginRequest` function. Under that function, we notice that another function, `UserDAO,` is called from a different file called `user-dao`. This function in the file handles the encryption process of credentials. So, find `validateLogin` under that file.

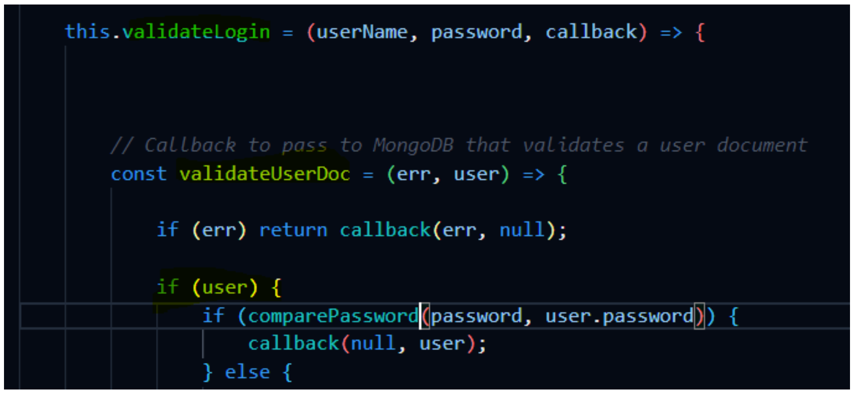
d. Ensure that `const bcrypt = require("bcrypt-nodejs"` is at the start of `user-dao.js`. Now, as `comparePassword is` asynchronous, we don't need this helper function anymore:



```
    this.validateLogin = (userName, password, callback) => {

        // Helper function to compare passwords
        const comparePassword = (fromDB, fromUser) => {
            return fromDB === fromUser;
            /*
            // Fix for A2-Broken Auth
            // compares decrypted password stored in this.addUser()
            return bcrypt.compareSync(fromDB, fromUser);
            */
        };

        // Callback to pass to MongoDB that validates a user document
        const validateUserDoc = (err, user) => {

```

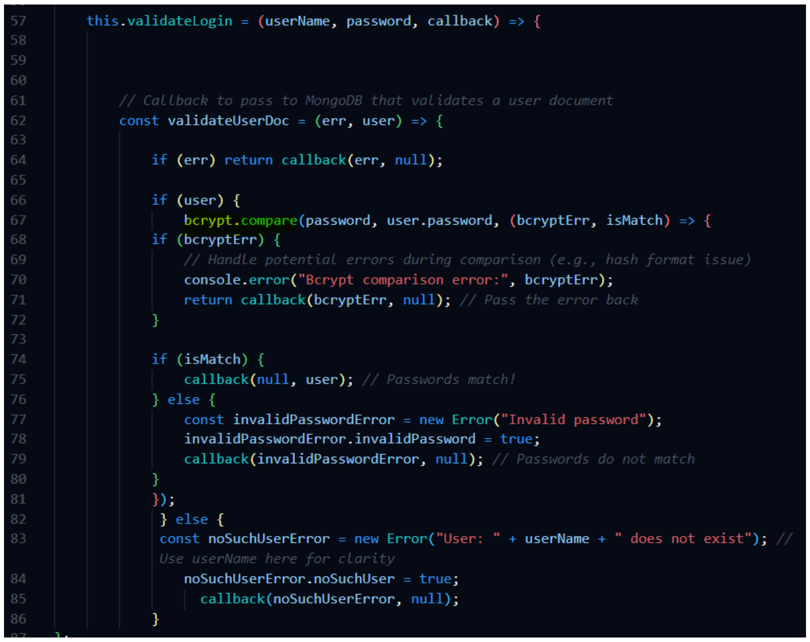
We can remove this function.

e. Now we will update `validateUserDoc` to use `bcrypt.Compare` asynchronously. Under the `validateUserDoc` function of the `validateLogin` method, find `if (user) {` block:



```
this.validateLogin = (userName, password, callback) => {



    // Callback to pass to MongoDB that validates a user document
    const validateUserDoc = (err, user) => {

        if (err) return callback(err, null);

        if (user) {
            if (comparePassword(password, user.password)) {
                callback(null, user);
            } else {
```

f. We will replace this function with another `bcrypt.compare` function:



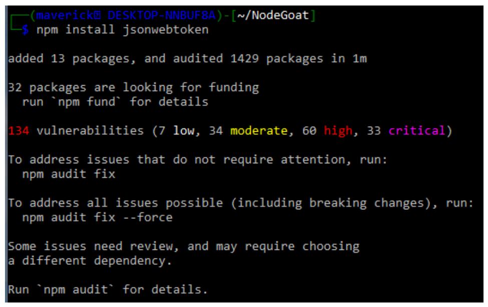
```
    this.validateLogin = (userName, password, callback) => {



        // Callback to pass to MongoDB that validates a user document
        const validateUserDoc = (err, user) => {

            if (err) return callback(err, null);

            if (user) {
                bcrypt.compare(password, user.password, (bcryptErr, isMatch) => {
            if (bcryptErr) {
                // Handle potential errors during comparison (e.g., hash format issue)
                console.error("Bcrypt comparison error:", bcryptErr);
                return callback(bcryptErr, null); // Pass the error back
            }

            if (isMatch) {
                callback(null, user); // Passwords match!
            } else {
                const invalidPasswordError = new Error("Invalid password");
                invalidPasswordError.invalidPassword = true;
                callback(invalidPasswordError, null); // Passwords do not match
            }
            });
             } else {
             const noSuchUserError = new Error("User: " + userName + " does not exist"); // Use userName here for clarity
                noSuchUserError.noSuchUser = true;
                  callback(noSuchUserError, null);
            }
```

With this, we have completed the **hashing of passwords** on the login and signup steps.

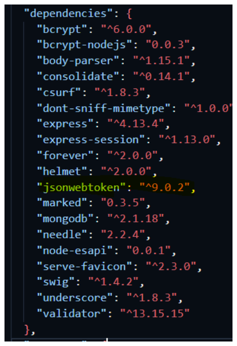
3- Implementing JWT Authentication:

JWTs (JSON Web Tokens) are a modern way to handle session management, providing a stateless and scalable alternative to traditional server-side sessions. So will implement this technology in our application's authentication system.

E. **Install `jsonwebtoken`:**
   a. With `npm install jsonwebtoken`, add JWT into the nodegoat dependency:



```
┌──(maverick㉿ DESKTOP-NNBUF8A)-[~/NodeGoat]
└─$ npm install jsonwebtoken

added 13 packages, and audited 1429 packages in 1m

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

134 vulnerabilities (7 low, 34 moderate, 60 high, 33 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues possible (including breaking changes), run:
  npm audit fix --force

Some issues need review, and may require choosing
a different dependency.

Run `npm audit` for details.
```

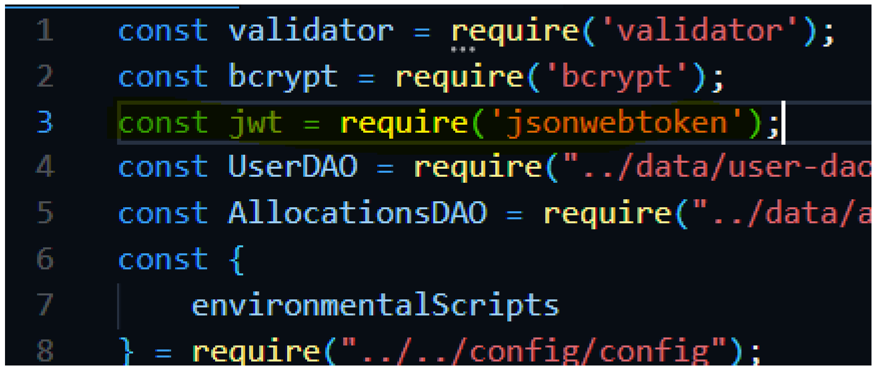


```
"dependencies": {
  "bcrypt": "^6.0.0",
  "bcrypt-nodejs": "0.0.3",
  "body-parser": "^1.15.1",
  "consolidate": "^0.14.1",
  "csurf": "^1.8.3",
  "dont-sniff-mimetype": "^1.0.0"
  "express": "^4.13.4",
  "express-session": "^1.13.0",
  "forever": "^2.0.0",
  "helmet": "^2.0.0",
  "jsonwebtoken": "^9.0.2",
  "marked": "0.3.5",
  "mongodb": "^2.1.18",
  "needle": "2.2.4",
  "node-esapi": "0.0.1",
  "serve-favicon": "^2.3.0",
  "swig": "^1.4.2",
  "underscore": "^1.8.3",
  "validator": "^13.15.15"
},
```

   b. Update and rebuild the Docker image:

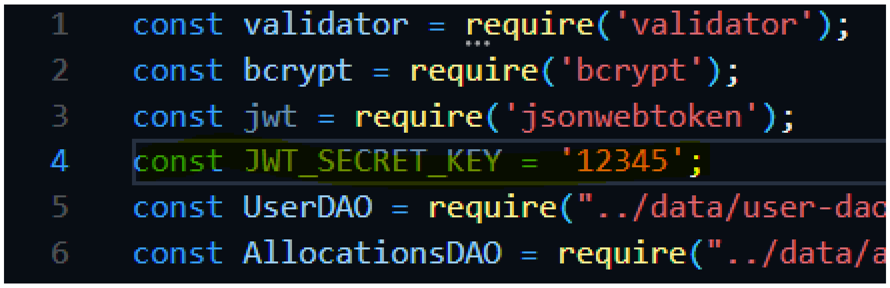
```
docker-compose build --no-cache
docker-compose up
```

F. **Update the code:**
   In this section, we will again modify `handleLoginRequest` in the `session.js` to issue a JWT instead of relying solely on `req.session.userId`.
   a. Add `jsonwebtoken` at the start of the code:



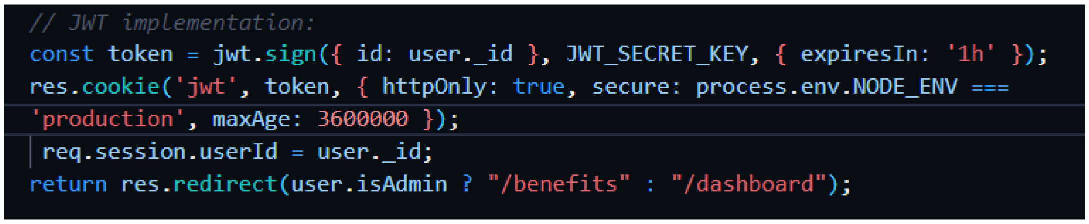
```
const validator = require('validator');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const UserDAO = require("../data/user-dac
const AllocationsDAO = require("../data/a
const {
    environmentalScripts
} = require("../../config/config");
```

b. We also need to add the secret key for future use:



```
const validator = require('validator');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const JWT_SECRET_KEY = '12345';
const UserDAO = require("../data/user-dao
const AllocationsDAO = require("../data/a
```

c. After the `validateLogin` function under `handleLoginRequest`, we will implement JWT authentication, using this code:



```
// JWT implementation:
const token = jwt.sign({ id: user._id }, JWT_SECRET_KEY, { expiresIn: '1h' });
res.cookie('jwt', token, { httpOnly: true, secure: process.env.NODE_ENV ===
'production', maxAge: 3600000 });
 req.session.userId = user._id;
return res.redirect(user.isAdmin ? "/benefits" : "/dashboard");
```

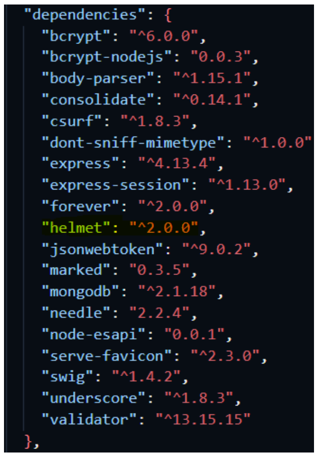
Adding this simple code will provide a stateless and scalable alternative to traditional server-side sessions.

4- Securing HTTP Headers:

A helmet helps secure your Express app by setting various HTTP headers. So in this last section, we will secure the HTTP header of NodeGoat.

G. **Install a `helmet`:**

a. With `npm install helmet,` add JWT into the nodegoat dependency:



```
"dependencies": {
  "bcrypt": "^6.0.0",
  "bcrypt-nodejs": "0.0.3",
  "body-parser": "^1.15.1",
  "consolidate": "^0.14.1",
  "csurf": "^1.8.3",
  "dont-sniff-mimetype": "^1.0.0"
  "express": "^4.13.4",
  "express-session": "^1.13.0",
  "forever": "^2.0.0",
  "helmet": "^2.0.0",
  "jsonwebtoken": "^9.0.2",
  "marked": "0.3.5",
  "mongodb": "^2.1.18",
  "needle": "2.2.4",
  "node-esapi": "0.0.1",
  "serve-favicon": "^2.3.0",
  "swig": "^1.4.2",
  "underscore": "^1.8.3",
  "validator": "^13.15.15"
},
```

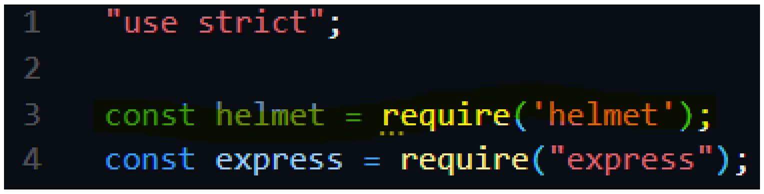
b. Update and rebuild the Docker image to reflect the changes:

```
docker-compose build --no-cache
docker-compose up
```

H. **Update the code:**

For this task, we will be using server.js and adding a few lines under it.

a. Add `helmet` at the start of the code:



```
"use strict";

const helmet = require('helmet');
const express = require("express");
```

b. Add `app.use(helmet());` in the middleware of the code.

With this, now the server uses Helmet to secure an HTTP header.

Submitted by ***Athar Imran***