Lab 10

Re-submit Assignment

Due Nov 21 by 11:59pm **Points** 100 **Submitting** a file upload

CS-546 Lab 10

Authentication and Middleware

For this lab, we will creating a basic server with user login, based on a static JSON file with usernames and **hashed** passwords. In addition, we will be creating a very basic logging middleware.

We will be using two new npm packages for this lab:

- **bcrypt (https://www.npmjs.com/package/bcrypt)**: a password hasing library. If you have problems installing that modules (since it uses C++ bindings), you can also try **bcrypt.js (https://www.npmjs.com/package/bcryptjs)**, which has the same API but is written in 100% JS.
- **express-session (https://www.npmjs.com/package/express-session)**: a simple session middleware for Express.

Routes

GET `/`

The root route of the application will do one of two things:

1. If the user is authenticated, it will redirect to `/private`.
2. If the user is not authenticaed, it will render a view with a login form for a username and password. The form used to submit the POST request to the server **must** have an `id` of `login-form`. The input for the username must have a `name`/`id` of `username`; the input for the password must have `name`/`id` of `password`.

An authenticated user should not ever see the login screen.

POST `/login`

This route is simple: making a POST to this route will attempt to log a user in with the credentials they provide in the login form.

If the user provides a successful username / password combination, you will set a cookie named `AuthCookie`. This cookie must be named `AuthCookie` or your assignment will receive a major point deduction. After logging in, you will redirect the user to the `/private` route.

If the user does **not** provide a valid login, you will render the login screen once again, and this time show an error message (along with an HTTP 401 status code) to the user explaining that they did not provide a valid username and/or password.

GET `/private`

This route will be simple, as well. This route will be protected your own authentication middleware to only allow valid, logged in users to see this page.

If the user is logged in, you will make a simple view that displays all details **except** the password for the currently logged in user.

Also, you will need to have a hyperlink at the bottom of the page to `/logout`.

GET `/logout`

This route will expire/delete the `AuthCookie` and inform the user that they have been logged out. It will provide a URL hyperlink to the `/` route.

User Data

You will use the following information to compose your users. **For the sake of this assignment and focusing on authentication, you will store them in a file and not in MongoDB**.

For example, you may create a `users.js` file with the following information:

```
module.exports = [
  {
    _id: 0,
    username: "masterdetective123",
    hashedPassword: "<The Hash Here>",
    firstName: "Sherlock",
    lastName: "Holmes"
  }, // etc, dont forget the other data
  {
    _id: 1,
    username: "lemon",
    hashedPassword: "<The Hash Here>",
    firstName: "Elizabeth",
    lastName: "Lemon"
  } // etc, dont forget the other data
];
```

Remember, all passwords must be hashed at all times using bcrypt!

You **do not** need to create a signup form for users! Simply add these users, with any associated data you may need, with **hashed** passwords to an array in memory. You can assign them different arbitrary IDs, as long as they are **unique**.

For the sake of simplicity of the assignment, I have supplied you with pre-hashed passwords through 16 salt rounds. You may hardcode the hashes, but not the actual passwords, in your data modules. **You will lose points for passwords found within your codebase, even in comments.** The passwords listed below are the passwords you will input into the login form that need to work.

User 1: Sherlock Holmes

Username: masterdetective123

First Name: Sherlock

Last Name: Holmes

Profession: Detective

Bio: Sherlock Holmes (/ˈʃɜːrlɒk ˈhoʊmz/) is a fictional private detective created by British author Sir Arthur Conan Doyle. Known as a "consulting detective" in the stories, Holmes is known for a proficiency with observation, forensic science, and logical reasoning that borders on the fantastic, which he employs when investigating cases for a wide variety of clients, including Scotland Yard.

Password: elementarymydearwatson

Hashed Password: `$2a$16$7JKSiEmoP3GNDSalogqgPu0sUbwder7CAN/5wnvCWe6xCKAKwlTD.`

(Note the `.` character at the end)

User 2: Liz Lemon

Username: lemon

First Name: Elizabeth

Last Name: Lemon

Profession: Writer

Bio: Elizabeth Miervaldis "Liz" Lemon is the main character of the American television series 30 Rock. She created and writes for the fictional comedy-sketch show The Girlie Show or TGS with Tracy Jordan.

Password: damnyoujackdonaghy

Hashed Password: `$2a$16$SsR2TGPD24nfBpyRlBzINeGU61AH0Yo/CbgfOlU1ajpjnPuiQaiDm`

User 3: Harry Potter

Username: theboywholived

First Name: Harry

Last Name: Potter

Profession: Student

Bio: Harry Potter is a series of fantasy novels written by British author J. K. Rowling. The novels chronicle the life of a young wizard, Harry Potter, and his friends Hermione Granger and Ron Weasley, all of whom are students at Hogwarts School of Witchcraft and Wizardry . The main story arc concerns Harry's struggle against Lord Voldemort, a dark wizard who intends to become immortal, overthrow the wizard governing body known as the Ministry of Magic, and subjugate all wizards and Muggles.

Password: quidditch

Hashed Password: `$2a$16$4o0WWtrq.ZefEmEbijNCGukCezqWTqz1VWlPm/xnaLM8d3WlS5pnK`

Using `express-session`

This middleware package does one (fairly simple) thing. It creates a cookie for the browser that will be used to track the current session of the user, after we verify their login. We will expand on the `req.session` field to store information about the currently logged in user. You can see an example using `req.session` **here (https://github.com/expressjs/session#reqsession)** .

To initialize the middleware, you must do the following:

```
// Your app.js file

const session = require('express-session')

...

app.use(session({
  name: 'AuthCookie',
  secret: 'some secret string!',
  resave: false,
  saveUninitialized: true
}))
```

You can read more about session's different configuration options **here (https://github.com/expressjs/session#options)** . For the sake of this lab, the above configuration is all you will need.

Authentication Middleware

This middleware will `only` be used for the GET `/private` route and will do one of the following:

1. If a user is not logged in, you will return an HTML page saying that the user is not logged in, and the page must issue an HTTP status code of `403`.
2. If the user is logged in, the middleware will "fall through" to the next route calling the `next()` callback.

See **this reference (https://expressjs.com/en/guide/writing-middleware.html)** in the express documentation to read more about middleware.

Logging Middleware

This middleware will log to your console for every request made to the server, with the following information:

- Current Timestamp: `new Date().toUTCString()`
- Request Method: `req.method`
- Request Route: `req.originalUrl`
- Some string/boolean stating if a user is authenticated

There is no precise format you must follow for this. The only requirement is that it logs the data stated above.

An example would be:

```
[Sun, 14 Apr 2019 23:56:06 GMT]: GET / (Non-Authenticated User)
[Sun, 14 Apr 2019 23:56:14 GMT]: POST /login (Non-Authenticated User)
[Sun, 14 Apr 2019 23:56:19 GMT]: GET /private (Authenticated User)
[Sun, 14 Apr 2019 23:56:44 GMT]: GET / (Authenticated User)
```

Requirements

1. All previous lab requirements still apply.
2. You must remember to update your package.json file to set app.js as your starting script!
3. **Your HTML must be valid (https://validator.w3.org/#validate_by_input)** or you will lose points on the assignment.
4. Your HTML must make semantical sense; usage of tags for the purpose of simply changing the style of elements (such as i, b, font, center, etc) will result in points being deducted; think in terms of content first, then style with your CSS.
5. You can be as creative as you'd like to fulfill front-end requirements; if an implementation is not explicitly stated, however you go about it is fine (provided the HTML is valid and semantical). Design is not a factor in this course.
6. All inputs must be properly labeled!