Lab 9

Submit Assignment

Due Sunday by 11:59pm **Points** 100 **Submitting** a file upload **File Types** zip

CS-546 Lab 9

A Simple User Login System

Using the concepts learned on authentication, middleware, etc, you will use a node package to implement a basic user system with only 3 routes.

This lab will require a good deal of reading, but relatively little code to be written out.

You must first read up on **Passport** (http://passportjs.org/docs) and what it accomplishes, then read up on the **Passport Local Strategy** (https://github.com/jaredhanson/passport-local) in order to complete this lab.

Passport is a very simple, easy to drop in middleware that allows you to quickly add application authentication.

A strategy is a way of checking whether or not a request is authenticated.

Your 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 authenticated, it will render a view with a login screen for a username and password. If there are any errors from a previous login, it will display the errors as well.

POST /login

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

You will check the provided username and password by telling passport to use a LocalStrategy that calls your data modules to get a user by username and password.

If the user cannot be authenticated, they will be redirected to the `/` route and an error message must be displayed.

If the user has been authenticated, they will be redirected to `/private` and details will be shown about the user.

GET /private

This route will be simple, as well. You will protect the `/private` route with passport to only allow validated users to login.

When logged in, passport provides you access to the `user` property on your `request` data in express. Using the `req.user` property, you will make a simple view that displays all details **except** the password for the currently logged in user.

Users

You will use the following information to compose your users. **For the sake of this assignment and focusing on authentiction, you will store them in memory and not in a database**; the data module methods you create to access and search for your users must, however, return promises.

For example, you may store, in your `users.js` file, something like this:

```
const users = [
  { _id: "1245325124124", username: "masterdetective123", hashedPassword: "     $2a$06$PHPllli6U0D0aXaKMVejxOfvwYISspU7kumJ.l0z
  { _id: "723445325124124", username: "lemon", hashedPassword: "$2a$06$SagJO.YW8T7c7Fzh.0VaIuYaAetQKsU2PbmI.VzzjjfWKA8yyLbQe",
]
```

Remember, all passwords must be hashed at all times using an algorithm such as 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.

For the sake of simplicity of the assignment, you may use an online bcrypt hasher in order to generate your passwords, or you may run the bcrypt node module locally and generate them and hardcode the hashes into your user objects. 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

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

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

Requirements

1. You **must not submit** your node_modules folder
2. You **must remember** to save your dependencies to your package.json folder
3. You must do basic error checking in each function
    1. Check for arguments existing and of proper type.
    2. Throw if anything is out of bounds (ie, trying to perform an incalculable math operation or accessing data that does not exist)
    3. If a function should return a promise, instead of throwing you should return a rejected promise.
4. You **must remember** to update your package.json file to set `app.js` as your starting script!
5. **Your HTML must be valid** **(https://validator.w3.org/#validate_by_input)** or you will lose points on the assignment.
6. 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.
7. **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.
8. **Your client side JavaScript must be in its own file and referenced from the HTML accordingly**.
9. You must pass accessibility tests.
10. You must plug in all common security issues!
11. You must error-check all forms for common errors.