Using express-session for login

To implement a simple session-based login in your Express app you'll need the following modules:

- express-session (`npm i express-session`)
- crypto (built-in to Node.js so you don't need to install anything)
    - This contains the scryptSync() (or scrypt()) function we'll use to hash the password.

As we'll be reading from MongoDB, we'll be using Mongoose for the various functionality. Some code will not be include to keep this document short but the relevant code will be included.

There are a couple environment variables we'll use:

- DBUSER
- DBPWD
- DBHOST
- DBNAME
- SESSIONSECRET (set to some random string – to be used for hashing the session)
- SALT (a random string at least 16 bytes long to be used for hashing the user password)

To enable and create sessions (you can put this in *index.js* if you wish):

```
const sessions = require("express-session");
…
app.use(
  sessions({
    secret: process.env.SESSIONSECRET,
    name: "MyUniqueSessID",
    saveUninitialized: false,
    resave: false
  })
);
```

`saveUnitialized: false` is useful for logins because there may be some pages where the user doesn't need to be logged in. Setting this to false means that if a session is not initialized, it won't be saved to the store.

`resave: false` will ensure the session is only saved if it's modified.

<u>For the login template:</u>

```
block layout-content
  .main-content
    h1.page-title Login
    if err
      p #{err}
    form(method="post", action="/user/login")
      div
        label(for="u") Username:
```

```
        input(type="text", id="u", name="u")
      div
        label(for="pw") Password:
        input(type="password", id="pw", name="pw")
      button(type="submit") Log in
```

For the User model/Mongoose stuff (in its own file):

```
const mongoose = require("mongoose");

const dbUrl =
`mongodb+srv://${process.env.DBUSER}:${process.env.DBPWD}@${process.env.DBHOST}/${pro
cess.env.DBNAME}`;

async function connect() {
  await mongoose.connect(dbUrl); //connect to mongodb
}

const { scryptSync } = require("crypto");

const db = require("../../db"); //shared db stuff

const UserSchema = new mongoose.Schema({
  user: String,
  password: String
});
const User = mongoose.model("User", UserSchema);

async function authenticateUser(username, pw) {
  await db.connect();
  let key = scryptSync(pw, process.env.SALT, 64);
  //check for existing user with matching hashed password
  let result = await User.findOne({
    user: username,
    password: key.toString("base64")
  });
  if (result)
    return true;
  else
    return false;
}
async function getUser(username) {
  await db.connect();
  //just check if username exists already
  let result = await User.findOne({ user: username });
  /* if (result) {
    return result;
  }
  return false; */
  return (result) ? result : false;
}
async function addUser(username, pw) {
  await db.connect();
  //add user if username doesn't exist
  let user = await getUser(username);
  console.log(user);
  if (!user) {
    let key = scryptSync(pw, process.env.SALT, 64);
    let newUser = new User({
      user: username,
      password: key.toString("base64")
    });
    let result = await newUser.save();
    if (result === newUser)
      return true;
    else
      return false;
  } else {
    return false;
  }
}

module.exports = {
  authenticateUser,
  getUser,
  addUser
}
```

The above code has two important functions for the login:

- authenticateUser(u, pw)
  - Returns true if user and hashed password matches a document in the database.
  - scryptSync() is a synchronous function used to create a key. The password value (from the login form), the salt value (for hashing), and the length (64) are defined. To actually get the hash, you'll take the key then use toString("base64") to get the Base64 value.
- getUser(username)
  - Returns the user from the DB if the username exists. Returns false if no such user exists.
  - You can use this for retrieving user data or during a registration process to ensure a username doesn't already exist.

For the actual logic/route code (assuming /user can only be viewed by logged-in users and Mongoose model/functions have been imported from a file into **userModel**):

```
// /user
app.get("/", async (request, response) => {
  //get user from session and render user page
  if (request.session.loggedIn) {

    response.render("user/user", { username: request.session.user });
  } else {
    //if session variable doesn't exist redirect to /user/login
    response.redirect("/user/login");
  }
});
```

```
// /user/login
app.get("/login", (request, response) => {

  response.render("user/login");
);

//login form submit
app.post("/login", async (request, response) => {

  //authenticate user and redirect to /user
  let auth = await userModel.authenticateUser(request.body.u, request.body.pw);
  console.log(auth);
  if (auth) {
    //if authenticated, set session variables
    request.session.loggedIn = true;
    request.session.user = request.body.u;
    //now redirect to /user
    response.redirect("/user");
  } else {
    response.render("user/login", { err: "User not found" });
  }
});

// /user/logout
app.get("/logout", (request, response) => {

  //destroy session and redirect to home
  request.session.destroy();
  response.redirect("/");
});
```

For any pages that require login, now all you need to do is just check if request.session.loggedIn is true. If it's not true, then redirect back to the login (see the callback function for /user).

For the logout, all that's needed to be done is to destroy the session, which will get rid of the values stored in the session (loggedIn and user which stored true if a user was logged in and the logged-in user's name, respectively).