User Authentication Part 1

Making a Basic User

Authentication So Far

- Up to this point, our authentication has been limited to having one password to secure parts of our site
- We've enhanced it such that we don't need to keep re-entering this password, thanks to sessions
- But most websites don't just have one password that gives everyone the same amount of access
- Typically a website has individual "users" who each have their own password that they made themselves

Storing the User

- Before we can create users, we'll need a place to store them
- A database is the best place for this, since we want users to stay between runs of the server
- We'll also want to be able to create new users without having to hard-code their information
- The schema for this would typically have some sort of unique login identifier (username, email) and a password to compare to when they login
- You could also have other information attached to users, but that would depend on what your app did

Storing the User (Sequelize Schema)

```
const User = sql.define("user", {
    id: {
        type: Sequelize.SERIAL,
        primaryKey: true,
        autoIncrement: true,
    },
    username: {
        type: Sequelize.STRING,
        notNull: true,
        unique: true,
    },
    password: {
        type: Sequelize.STRING,
        notNull: true,
    },
});
```

Schema Notes

- We still have a separate ID column, rather than make the username the primary key, so that a user could change their name without breaking everything that references their ID
- We indicate that the username is unique, because otherwise, we wouldn't know which user was trying to login using which password
- Right now the password doesn't have anything special to it, but we'll end up adding more configuration to it later

Creating a User

- Creating one of these users is no different than anything else we've made in the database
- Simple create a POST form, add an Express `app.post()` route, and run `User.create()`
- You may also want to validate the inputs or ask for more data, but that's up to you

```
app.post("/signup", function(req, res) {
    User.create({
        username: req.body.username,
        password: req.body.password,
    })
    .then(function(user) {
        // Set their session to logged in, redirect to homepage
    })
    .catch(function(err) {
        // Re-render form with error message
    });
});
```

Logging a User In

- Logging a user in looks similar to creating one, by POSTing a form
- Except rather than adding something new to the database, we'll be searching for an existing entry
- We can use Sequelize's `findOne()` method to search for the user, and specify which user we're looking for with a where clause
- We **cannot** use findById because users don't enter their numeric ID primary key, they enter a human-readable username

Logging a User In (Example)

```
app.post("/login", function(req, res) {
    User.findOne({
        where: {
            username: req.body.username,
        },
    })
    .then(function(user) {
        if (user) {
            // Log them in
        }
        else {
            // Re-render form with "invalid user" error message
        }
    })
    .catch(function(err) {
        // Re-render form with database error message
    });
});
```

Validating The User

- But hold up, we can't just log them in because they *say* they're that user
- We'll need to compare the password they provided, with the password we stored
- For now, we'll do a simple javascript comparison between the two

```
User.findOne(/* ... */).then(function(user) {
    if (user) {
        if (user.password === req.body.password) {
            // Log them in
        }
        else {
            // Re-render form with "invalid password" error message
        }
    }
});
```

Adding the User to the Session

- Now that we know the user is who they say they are, we can log them in
- The best way to do that is to add their user info to `req.session`
- That way any request they make will come with `req.session.user`

```
if (user.password === req.body.password) {
    req.session.user = user;
    res.redirect("/home");
}
```

"Using" the User

- And there you have it, your user is now available on all new requests
  - A middleware could redirect requests without `req.session.user` to a login form
  - You could also only allow certain pages to be shown to certain users
  - Your template can now display things specific to being logged in, or with your user's information
- The user can also be logged out just as easily by nulling out `req.session.user` on another request

Issues With Our Implementation

- Despite working, there are a few things too naive about our users implementation:
  - The session can't hold the advanced User instance object, only the basic data
  - If we changed data about our user, our session user object would no longer match our database
  - Our passwords are stored in plain text, completely vulnerable to a hacker
  - Our sessions only last for as long as the server is up, so restarting the server requires us to log back in
- We'll be addressing these issues in future lessons

Challenge: Enhance Our App

- So now that we've got basic user authentication, let's build on that
- In order, try to accomplish as many of these features as you can:
  - Add a confirm password input when signing up. This should be validated server-side, making sure that both passwords were provided and the same.
  - Add a check to the "/home" route to make sure you're logged in, and redirect you to login if you're not
  - Add a logout route and link to the "/home" page that logs you out
- Feel free to catch up on your homework once you've accomplised all 3 goals