Passwords

Here's a scenario: you're on a site and you request a password reset. The website then just sends you your current password. If this happens, you should NEVER use that password again (or the site, either, if you can help it).

- Best practice (honestly, only practice at this point) is to store passwords in your database as hashed values.
- You can't avoid sending the password to the server as plaintext. You could hash it on the frontend, but that's mostly pointless as the end user has access to all of the frontend code anyway. This means that they could inspect the JavaScript, find the hashing function, and just send their own pre-hashed password while ignoring our hashing function. This would be an issue, because our server doesn't know if the hash came from the actual password, or if it was generated by an attacker.

What is Hashing?

- Hashing is basically just taking a value (in this case, a password) and transforming it into a random set of characters using an algorithm.
- Hashing is one way and one way only. Once it's changed, you can't change it back. This means that if you take a certain value, it will always change to the same value.
- Is this secure? Mostly, however it's only secure for a set amount of time. Exactly how long is dependant on the attacker. Basically, any hash can be cracked eventually, but the amount of time it would require to get it right is so ludicrously long that it just doesn't make sense unless you have access to a super computer. For most malicious actors, it's not worth their time to try and guess the plaintext version of a hashed password.
- Because of this, passwords are converted to a hashed value and then stored in the database. When a plaintext password is provided via a login request or equivalent, the order of operations is this:
    1. Plaintext password is taken from `req` and turned into a hashed value.
    2. User with the username provided is found in the database and the saved hashed password is retrieved.
    3. The two hashed values (one from the database one form the `req`) are compared. If they are equal, the user is allowed access. If they are not equal, a wrong username/password message is sent to the front end.

How to Hash

- The `argon2` package is the most preferred method for hashing. It's slower than the built in `crypto` package that Node has, but it is more secure (which is why it's slower). `npm install argon2`
- There are many more options that you can use with argon, but this is the most straight forward method.

    ```
    import argon2 from 'argon2';
    const hash = await argon.hash(password);
    ```

    That's it!
- Then you can simply compare the two with:

```
const verified = await argon.verify(plainTextPassword, hashedValuedFromDB);
```

'verified' will then be true or false depending on if they match.

See the docs for more info: https://www.npmjs.com/package/argon2