- Store with a nonce, store the hashes
- Generate a random salt per user
- Once we create a hash using the nonce then we should validate it to make sure everything went to plan.
    - Can do it using prebuilt APIs.
- Two examples below using two different APIs.

Example 1 Node.js

- https://github.com/florianheinemann/password-hash-and-salt

```
$ npm install password-hash-and-salt --save
```

```
var password = require('password-hash-and-salt');

var myuser = [];

// Creating hash and salt
password('mysecret').hash(function(error, hash) {
  if(error)
    throw new Error('Something went wrong!');

  // Store hash (incl. algorithm, iterations, and salt)
  myuser.hash = hash;

  // Verifying a hash
  password('hack').verifyAgainst(myuser.hash, function(error, verified) {
    if(error)
      throw new Error('Something went wrong!');
    if(!verified) {
      console.log("Don't try! We got you!");
    } else {
      console.log("The secret is...");
    }
  });
})
```

- this creates a hash is of 270 characters length

Example 2 Node.js

- Uses the module bcrypt found below
  - https://www.npmjs.com/package/bcrypt

Create a hash:

```
var bcrypt = require('bcrypt');
var salt = bcrypt.genSaltSync(10);
var hash = bcrypt.hashSync("B4c0/\/", salt);
// Store hash in your password DB.
```

To check a password:

```
// Load hash from your password DB.
bcrypt.compareSync("B4c0/\/", hash); // true
bcrypt.compareSync("not_bacon", hash); // false
```

- Will need to change database type to BINARY(40) or if we don't care about space BINARY(60)
- CHAR(60) or VARCHAR(60) are exceptable as well.