Lab 2 Part 1

Password Hashing with Node.js

Overview

Data security is a big topic, that could take up a whole module by itself. The aim of this lab is therefore to encourage you to give consideration to the security of sensitive data when designing and developing data-driven applications, and also when communicating sensitive data over a network.

Learning Objectives

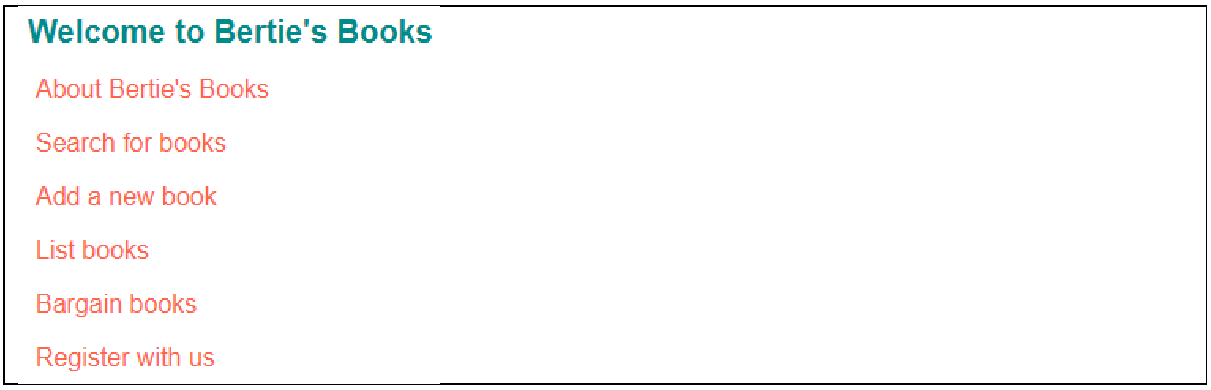
- Identify a Node.js module we can use to hash a password
- Describe how to compare a hashed password value (from a database) with an unhashed string (from a form)
- Using Git version control management

Tasks

Task 1: Set up your copy of the Berties Books web application

1. Set up the Berties Books application. You can base this on your work in last year's Dynamic Web Applications module, or you can download the code I have supplied on the VLE. On the VLE, there is application code and a database script for you to run on MySQL.

2. Once you have set up the application, test it to make sure all routes work. The home page should look something like this:


Welcome to Bertie's Books

About Bertie's Books

Search for books

Add a new book

List books

Bargain books

Register with us

Task 2: The security of data held in a database, working with hashed data

In the context of data-driven web application development, we have another kind of security concern to worry about: protecting data held in a database. In particular, we should worry about protecting sensitive data. A large part of this falls to the database and/or system administrator(s), but it is also a concern for middleware and frontend developers.

3. Install password hashing module (bcrypt) and use it in your web application

4. In order to install bcrypt module run:

```
npm install bcrypt
```

5. Edit your main.js file and then add the following lines of code related to '/registered' path (Do you know where to add these lines?) :

```
const bcrypt = require('bcrypt');
const saltRounds = 10;
const plainPassword = req.body.password;
```

6. Hash the password as follows before storing it in the database :

```
bcrypt.hash(plainPassword, saltRounds, function(err, hashedPassword) {
  // Store hashed password in your database.
})
```

7. Add a username and password field to the registration form.

8. Create a table that can store your user details in the database. You will need columns for the following: username, first name, last name, email and hashedPassword.

9. Add the piece of code to store the data in the database after the comment in the above code. Take a look at the `/bookadded` route if you need a hint as to how to do this.

10. Output the password and hashedPassword in the response. You can modify the response output to use the following code:

```
result = 'Hello '+ req.body.first + ' '+ req.body.last +' you are now
registered!  We will send an email to you at ' + req.body.email;
result += 'Your password is: '+ req.body.password +' and your hashed password
is: '+ hashedPassword;
res.send(result);
```

11. Save your `main.js` file and run your web application.

Note: Of course, the password and hashed passwords are displayed here only for debugging purposes. You would never display passwords in a web application!!

If you enter the same password for two users you will see that the hashed password would be different.

This is because we are using salt.

Task 5: Add '/listusers' route and page

12. Add a `/listusers` route and page to your web application. Do not include passwords on your `/listusers` page.

Refer to the `/list` route in your web application if you are not sure how to this task.

Task 6. Add a login page to your web application

13. Create a new login form to users and asks for username and password and create a new route `/login` to display it.

14. Create a new route `/loggedin` which will compare the collected form data with the data stored in the database and then display a message if login is successful and another message if login is not successful.

The `/loggedin` page is very similar to `/registered` page with one main difference: On the `/loggedin` page, you are not saving username and password in the database but comparing them with the username and password already saved in the database.

Comparing the passwords

In order to compare user's password collected from the login page with the hashed password saved in the database you will need to:

1. Select the hashed password for the user from the database
2. Compare the password supplied with the password retrieved from the database

For item 2, the following code template can be used:

```
  // Compare the password supplied with the password in the database
  bcrypt.compare(req.body.password, hashedPassword, function(err, result) {
    if (err) {
      // TODO: Handle error
    }
    else if (result == true) {
      // TODO: Send message
    }
    else {
      // TODO: Send message
    }
  });
```

Next week...

Next week you will be exporting your web application to GitLab and you will be submitting two URLs (Your web application URL and your Gitlab project URL).

END