

# WEB322 Assignment 8

## Submission Deadline:

Friday, August 11<sup>th</sup>, 2017 @ 11:59 PM

## Assessment Weight:

5% of your final course Grade

## Objective:

Enable users to update their credentials ("profile") using AJAX in a modal window & "harden" our web322-app by using one-way password encryption with bcrypt.

## Specification:

For this assignment, we will be removing all existing users from our web322-app and upgrading our DB to use **hashed passwords** for increased security (in a production environment, we cannot wipe out all users, however we're still in the development stage, and these are just test users). Additionally, we will be leveraging some jQuery and Bootstrap to make changing passwords a breeze for the end user.

**NOTE:** If you are unable to start this assignment because Assignment 7 was incomplete - email your professor for a clean version of the Assignment 7 files to start from (effectively removing any custom CSS or text added to your solution). Remember, you must successfully complete ALL assignments to pass this course.

## Getting Started / Adding bcryptjs:

To get started, once again open your web322-app folder in Visual Studio Code. If there were problems with previous assignments and things aren't working correctly, you can ask for a fresh copy of Assignment 7 (see above).

We will be using the "bcryptjs" 3<sup>rd</sup> party module, so we must go through the usual procedure to obtain it (and include it in our "data-service-auth.js" module).

1. Open the integrated terminal and enter the command: **npm install "bcryptjs" --save**
2. At the top of your **data-service-auth.js** file, add the line: **const bcrypt = require('bcryptjs');**

## Clearing out the "Users" collection

Since all of our new users will have encrypted (hashed) password, we will need to remove all of our existing test users. This can be done easily in **Robo 3T** (previously known as: **Robomongo**):

- **Using Robo 3T:** double-click the "users" collection to see all of the entries. Next, select them all, **right-click** and select **"Delete Documents..."** from the pop up menu.

It is also possible to programmatically delete all documents with a quick method call inside the "initialize" function in data-service-auth.js:

- **Using Mongoose to remove all Users:** Simply place the following code **immediately following** your `User = db.model("users", userSchema);` line inside the "initialize" function within your data-service-auth.js file:
  - `User.remove({ }, function (err) { });`
- **IMPORTANT NOTE!** You must remove this line once you have ran your server once and wiped out all the Users, otherwise **All Users** will be **wiped out** every time you **start your server!**

### Updating our data-service-auth.js functions to use bcrypt:

Now that we have the bcryptjs module included and our Users collection has been cleaned out, we can focus on updating the other two functions in our data-service-auth.js module. We will be using bcrypt to encrypt (hash) passwords in **registerUser(userData)** and validate user passwords against the encrypted passwords in **checkUser(userData)**:

#### Updating registerUser(userData)

- Recall from the Week 12 notes - to encrypt a value (ie: "myPassword123"), we can use the following code:
 

```
bcrypt.genSalt(10, function(err, salt) { // Generate a "salt" using 10 rounds
  bcrypt.hash("myPassword123", salt, function(err, hash) { // encrypt the password: "myPassword123"
    // TODO: Store the resulting "hash" value in the DB
  });
});
```
- Use the above code to **replace** the user entered password (ie: **userData.password**) with it's **hashed version** (ie: **hash**) **before** continuing to save **userData** to the database and handling errors (from Assignment 7)
- If there was an error (ie, **if(err){ ... }**) trying to **generate the salt** or **hash the password**, **reject** the **returned promise** with the message "There was an error encrypting the password" and **do not** attempt to save **userData** to the database.

#### Updating checkUser(userData)

- Recall from the Week 12 notes - to compare an encrypted (hashed) value (ie: **hash**) with a plain text value (ie: **"myPassword123"**), we can use the following code:
 

```
bcrypt.compare("myPassword123", hash).then((res) => {
  // res === true if it matches and res === false if it does not match
});
```
- Use the above code to **verify** if the user entered password (ie: **userData.password**) matches the hashed version for the requested user (**userData.user**) in the database (ie: **instead** of simply comparing `users[0].password == userData.password` as this will no longer work. The **compare** method must be used to compare the hashed value from the database to `userData.password`)
- If the passwords match (ie: **res === true**) **resolve** the returned promise without any message
- If the passwords do not match (ie: **res === false**) **reject** the returned promise with the message "Unable to find user: **user**" where **user** is the **userData.user** value

### Adding a new function "updatePassword(userData)":

- This function is very similar to our current registerUser(userData) function, except instead of saving **userData** to the database and handling errors, we will instead **update** the user, where the **user** is **userData.user** and set it's **password** value to the **hashed version** (see Week 8 Notes for more information on "update"):

```
User.update({ user: userData.user },
{ $set: { password: hash } },
{ multi: false })
.exec()
.then()
.catch();
```

- If the **update** method resolved successfully (ie, using .then()), **resolve** the returned promise without any message
- If the **update** failed (ie, using .catch()), **reject** the returned promise with the message: "There was an error updating the password for **user**" where **user** is the **userData.user** value

### Adding a new POST route "/api/updatePassword" to server.js:

Since we will be using AJAX for this route, we will need to **add** the line: **app.use(bodyParser.json());** at the top of our **server.js** file underneath the other **bodyParser** middleware function (app.use(bodyParser.urlencoded({ extended: true })));).

Once this is done, proceed to add the new POST route ("/api/updatePassword") alongside all of your other routes in server.js. The specification for this route is as follows:

- You can assume that **req.body** will contain the following JSON object:

```
{
  user: currentUser,
  currentPassword: currentPassword,
  password: newPassword,
  password2: newPasswordConfirm
}
```

where **currentUser**, is the logged in user, **currentPassword** is the logged in user's current password, **password** is what they're trying to change their password to, and **password2** is the confirmation (ie, password must be equal to password2)

- First, make a call to your data-service-auth.js **checkUser** function, to ensure that the **currentPassword** is correct, ie:

```
dataServiceAuth.checkUser({ user: req.body.user, password: req.body.currentPassword })
```

- If the **checkUser** promise is **rejected** send the following JSON back to the client: {errorMessage: **err**} where **err** is the error returned from the **checkUser** promise

- If the **checkUser** promise **resolves** successfully, make a call to your data-service-auth.js **updatePassword** with the data in **req.body**, ie:

```
dataServiceAuth.updatePassword(req.body)
```

- If the **updatePassword** promise is **rejected** send the following JSON back to the client: {errorMessage: **err**} where **err** is the error returned from the **updatePassword** promise
- If the **updatePassword** promise **resolves** successfully, send the following JSON back to the client: {successMessage: "Password changed successfully for user: **user**"} where **user** is the **req.body.user** value

## Adding Client Side JavaScript

In order for us to be able to make AJAX calls and handle the returned data, we will need some client-side JavaScript:

- In your "public" folder, create a new "js" folder (you should now have two folders: "css" and "js")
- Inside the js folder, create a new file: **main.js**
- Inside your **layout.hbs** file, include this file in the <head>...</head> element. The src path should be **"/js/main.js"**
- Since there is only a single week to complete this assignment, we will provide the code for the **entire file** (below), however it is your job to **comment the code** with as much detail as possible:

```
function hidePasswordMessages(){
  $("#passwordChangeSuccess").addClass("hide");
  $("#passwordChangeError").addClass("hide");
}

function requestPasswordChange(username) {
  $.ajax({
    url: "/api/updatePassword",
    type: "POST",
    data: JSON.stringify({
      user: username,
      currentPassword: $("#currentPassword").val(),
      password: $("#password").val(),
      password2: $("#password2").val(),
    }),
    contentType: "application/json"
  })
  .done(function (data) {
    hidePasswordMessages();
    if(data.successMessage){
      $("#passwordChangeSuccess").removeClass("hide").children(".alert").text(data.successMessage);
    }else if(data.errorMessage){
      $("#passwordChangeError").removeClass("hide").children(".alert").text(data.errorMessage);
    }
  })
  .fail(function (jqXHR) {
    $("#passwordChangeError").removeClass("hide").children(".alert").text("AJAX Error: " + jqXHR.responseText);
  });
}
```

## Creating a "User Dropdown"

- To begin: **remove the existing Log Out link** - it should look something like:

- In it's place, use the example from the Week 11 notes for "Dropdown Buttons" (copy / paste)
- Instead of the text "Dropdown" in the "btn-primary" button, use the following:
  - The "user" glyphicon followed by the currently logged in user name (ie: session.user.username)
- Your dropdown should only contain 2 links with a "separator" between them (instead of 4 links and a separator between link 3 & 4, in the example):
  - The first (1<sup>st</sup>) link will have the "user" glyphicon followed by the text "Profile". This link will link to "#" but have the attributes: **data-toggle="modal"** and **data-target="#profileModal"**, which ensures that clicking on the link will open our "profileModal"
  - The next item will be a "separator"
  - The second (2<sup>nd</sup>) link will have the "off" glyphicon followed by the text "Log Out". This link will link to "/logout" as before
- When your user dropdown is complete, it should appear as the following (when expanded - assuming the logged in user is "asdf"):



As you can see from the code above, it is clear that we have a modal window with an id: "profileModal". The skeleton of this modal window is provided below, followed by some guidelines on what elements must be added to the modal and where to include the code.

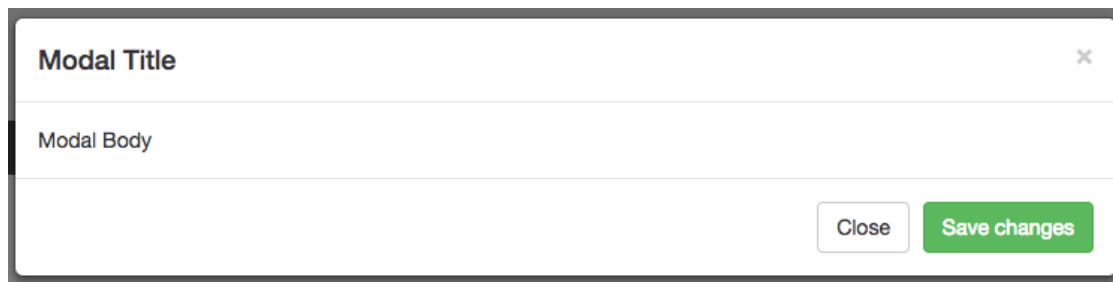
- To begin, copy the below code and paste it at the **bottom** of your **layout.hbs** file, **just before </body>**. The idea is that it's always the last element on the page when the user is **logged in**:

```

{{#if session.user}}
<!-- Profile Modal -->
<div class="modal" id="profileModal" data-backdrop="static" tabindex="-1" role="dialog" aria-labelledby="profileModalTitle">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <form onsubmit="requestPasswordChange('{{session.user.username}}'); return false;">
        <div class="modal-header">
          <button type="button" class="close" data-dismiss="modal" aria-label="Close"><span aria-
hidden="true">&times;</span></button>
          <h4 class="modal-title" id="profileModalTitle">Modal Title</h4>
        </div>
        <div class="modal-body">
          <div class="row">
            <div class="col-md-12">
              Modal Body
            </div>
          </div>
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
          <button type="submit" class="btn btn-success">Save changes</button>
        </div>
      </form>
    </div>
  </div>
</div>
{{/if}}

```

- If you click on "Profile" in the user dropdown, your modal should appear as:



- To complete this modal, ensure that you make the following updates:
- "Modal Title" should instead be "Profile"
- Inside the "modal-body" include the following form elements:

Input type	Properties
text	<ul style="list-style-type: none"> <li>disabled</li> <li>class="form-control"</li> <li>value= <i>currently logged in user's username, ie: <b>session.user.username</b></i></li> </ul>

password	<ul style="list-style-type: none"> <li>• class="passwordReset form-control"</li> <li>• id="currentPassword"</li> <li>• placeholder="Current Password"</li> </ul>
password	<ul style="list-style-type: none"> <li>• class="passwordReset form-control"</li> <li>• id="password"</li> <li>• placeholder="New Password"</li> </ul>
password	<ul style="list-style-type: none"> <li>• class="passwordReset form-control"</li> <li>• id="password2"</li> <li>• placeholder="Confirm Password"</li> </ul>

- Inside the "modal-body" include the following "error" column (note: this does not have to be a column, if you want to try a different design, it just needs the class="hide" and the id="passwordChangeError":

```
<div class="col-md-12 hide" id="passwordChangeError">
  <div class="alert alert-danger"></div>
</div>
```

- Inside the "modal-body" include the following "success" column (note: this does not have to be a column, if you want to try a different design, it just needs the class="hide" and the id="passwordChangeSuccess":

```
<div class="col-md-12 hide" id="passwordChangeSuccess">
  <div class="alert alert-success"></div>
</div>
```

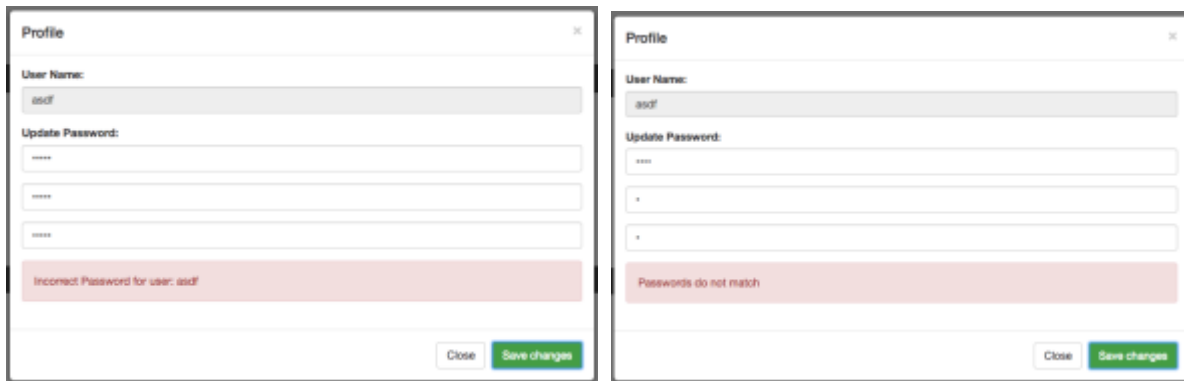
- When the modal window is complete, it should look something like this (if our current user is **asdf**):

## Testing the "Update Password:" functionality

To test the functionality, try the following situations:

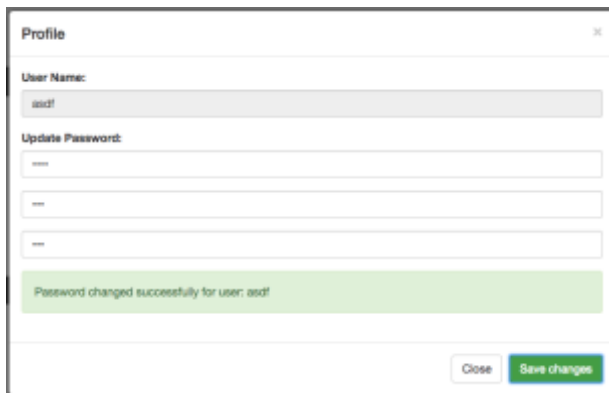
- "Current Password" doesn't match the password on record for the user

- The new and confirm passwords don't match up
- If either of the above scenarios are detected, you should see an error message in the modal depending on the type of error that was encountered, ie:



The image shows two side-by-side screenshots of a 'Profile' modal. Both modals have a title bar with 'Profile' and a close button. The first modal shows the 'User Name' field with 'asdf' and the 'Update Password' section with three password fields. A red error message at the bottom states 'Incorrect Password for user: asdf'. The second modal shows the same fields, but the red error message states 'Passwords do not match'. Both modals have 'Close' and 'Save changes' buttons at the bottom right.

- If everything is valid (ie: the "Current Password" is correct and the "New Password" is identical to the "Confirm Password", you should see a success message, ie:



The image shows a single screenshot of a 'Profile' modal. It has the same layout as the previous ones, with 'User Name' set to 'asdf' and three password fields in the 'Update Password' section. A green success message at the bottom states 'Password changed successfully for user: asdf'. The 'Close' and 'Save changes' buttons are at the bottom right.

## Sample Solution

To see a completed version of this app running, visit: <https://fast-spire-16915.herokuapp.com>



## Assignment Submission:

- Add the following declaration at the top of your server.js file:

```
/******  
* WEB322 – Assignment 08  
* I declare that this assignment is my own work in accordance with Seneca Academic Policy. No part of this  
* assignment has been copied manually or electronically from any other source (including web sites) or  
* distributed to other students.  
*  
* Name: _____ Student ID: _____ Date: _____  
*  
* Online (Heroku) Link: _____  
*  
*****/
```

- Publish your application on Heroku & test to ensure correctness
- Compress your web322-app folder and Submit your file to My.Seneca under **Assignments -> Assignment 8**

## Important Note:

- If the assignment will not run (using "node server.js") due to an error, the assignment will receive a **grade of zero (0)**.
- **NO LATE SUBMISSIONS** for assignments. Late assignment submissions will not be accepted and will receive a **grade of zero (0)**.
- After the end (11:59PM) of the due date, the assignment submission link on My.Seneca will no longer be available.