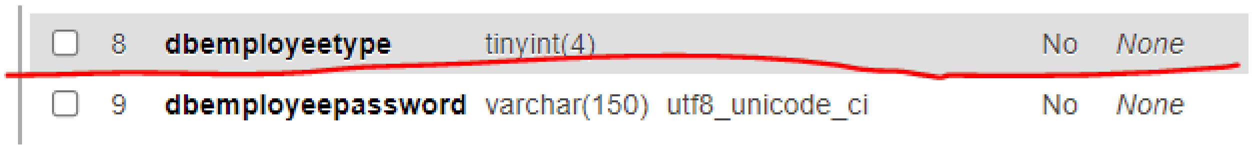
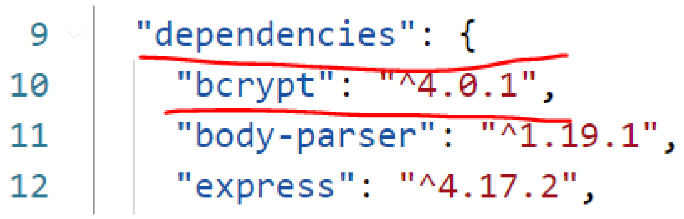
Lab 5

For this lab we will be adapting the previous pages for inserting and adding a field for passwords to the pages. In addition, a login page will be created for the site. You will need to adapt the database for the employee table to have the field added below for password.



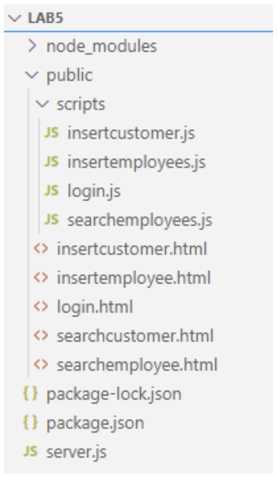
☐	8	**dbemployeetype**	tinyint(4)		No	*None*
☐	9	**dbemployeepassword**	varchar(150)	utf8_unicode_ci	No	*None*

From here copy all the files from Lab 4 into a folder called lastname_Lab5. Once the files are copied, open the lastname_lab5 folder in VS Code. Go into the package.json file and change the lab4 to lab5. Also, be sure to go into all the html pages and change the title from lab4 to lab5. Since we will be encrypting passwords, the library for bcrypt will need to be added. Add the code between the red lines below to the package.json file.



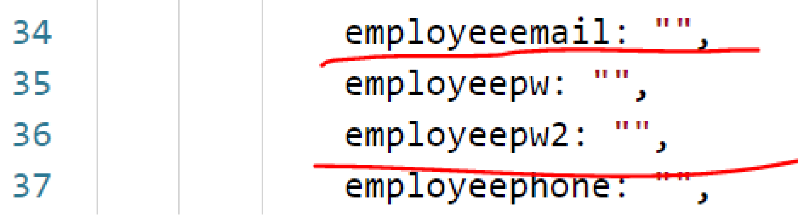
```
"dependencies": {
  "bcrypt": "^4.0.1",
  "body-parser": "^1.19.1",
  "express": "^4.17.2",
```

Once that is complete, you will need to run npm update in the terminal to install the libraries for bcrypt. There are 2 files in the dropbox for the lab, one for login.js and another for login.html Copy the login.js file into your scripts folder. Copy the login.html into the public folder. Your folder structure should look like the image below.



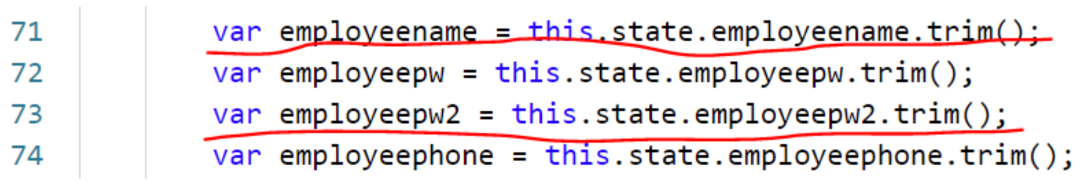
```
LAB5
  > node_modules
  v public
    v scripts
      insertcustomer.js
      insertemployees.js
      login.js
      searchemployees.js
    insertcustomer.html
    insertemployee.html
    login.html
    searchcustomer.html
    searchemployee.html
  package-lock.json
  package.json
  server.js
```

The first file that will be adapted is the insertemployee.js file. We need to add the fields for the password to be entered. There will need to be 2 fields for this, one to enter the password and one to confirm, so 2 variables will be needed, even though only one will be sent to the database. Add the 2 variables below to the getInitialState function in EmployeeForm2.



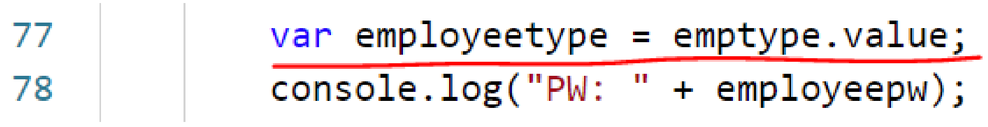
```
employeeemail: "",
employeepw: "",
employeepw2: "",
employeephone: "",
```

In the handleSubmit function, there will also need to be 2 variables created to hold the information after the Submit is clicked.



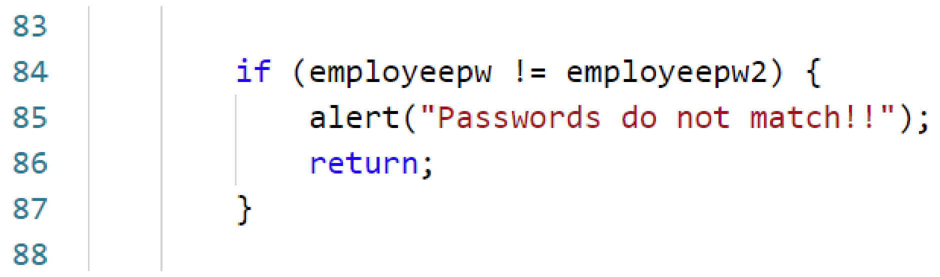
```
var employeename = this.state.employeename.trim();
var employeepw = this.state.employeepw.trim();
var employeepw2 = this.state.employeepw2.trim();
var employeephone = this.state.employeephone.trim();
```

For debugging purposes, the password is also sent to the console.log in the browser.



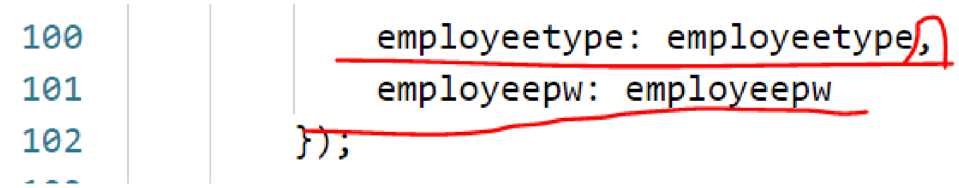
```
var employeetype = emptype.value;
console.log("PW: " + employeepw);
```

Where the if statements are for input validation, the program will need to check to ensure that the passwords match before being sent to the database. The code below will accomplish this.



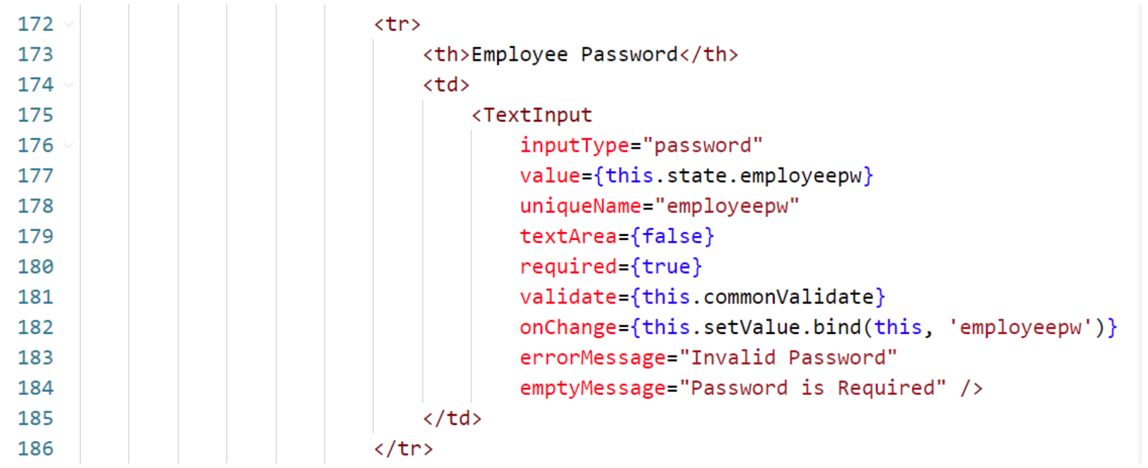
```

if (employeepw != employeepw2) {
    alert("Passwords do not match!!");
    return;
}

```

The information from the variables will need to be placed into the array that is sent to the server.js page, add the line of code below to add the variable to the array. In addition, remember to add the comma to the line above as well.



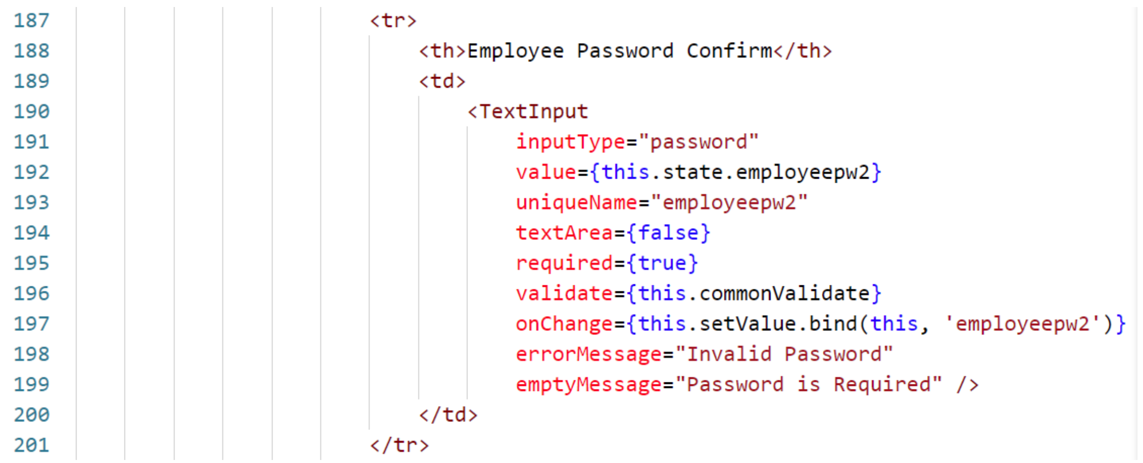
```
100             employeetype: employeetype,
101             employeepw: employeepw
102         });
```

Next, the fields on the form will need to be added for the passwords, after the code to add the field on the form for the email address, add the code below to add a field for the password. There is an additional line added we have not added previously. This is line 176, this will make the field a password type field, so the password cannot be read on the screen. We will add code below to make this functional in the TextInput area.



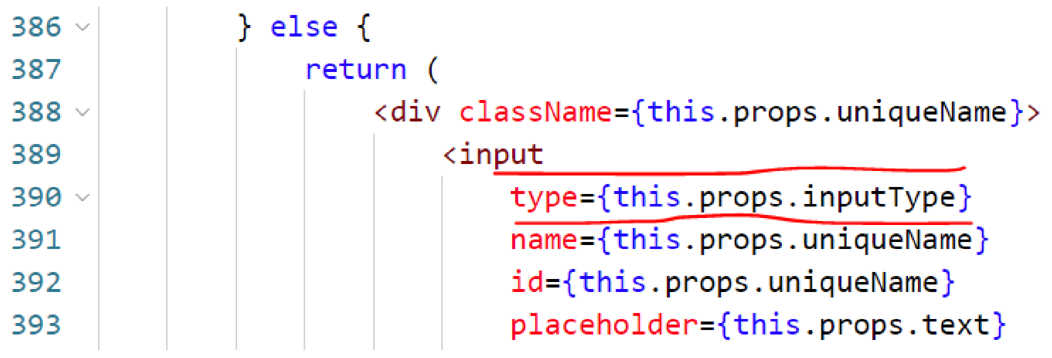
```
172                         <tr>
173                             <th>Employee Password</th>
174                             <td>
175                                 <TextInput
176                                     inputType="password"
177                                     value={this.state.employeepw}
178                                     uniqueName="employeepw"
179                                     textArea={false}
180                                     required={true}
181                                     validate={this.commonValidate}
182                                     onChange={this.setValue.bind(this, 'employeepw')}
183                                     errorMessage="Invalid Password"
184                                     emptyMessage="Password is Required" />
185                             </td>
186                         </tr>
```

Perform the same step to add a field for the confirm password.



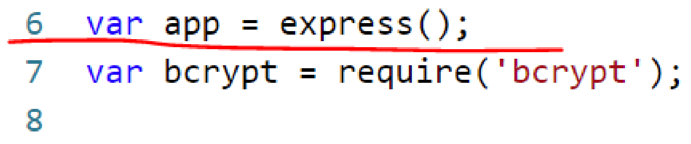
```
187                         <tr>
188                             <th>Employee Password Confirm</th>
189                             <td>
190                                 <TextInput
191                                     inputType="password"
192                                     value={this.state.employeepw2}
193                                     uniqueName="employeepw2"
194                                     textArea={false}
195                                     required={true}
196                                     validate={this.commonValidate}
197                                     onChange={this.setValue.bind(this, 'employeepw2')}
198                                     errorMessage="Invalid Password"
199                                     emptyMessage="Password is Required" />
200                             </td>
201                         </tr>
```

Add the line of code between the red lines below to allow the type option we added above to be functional in the TextInput field.



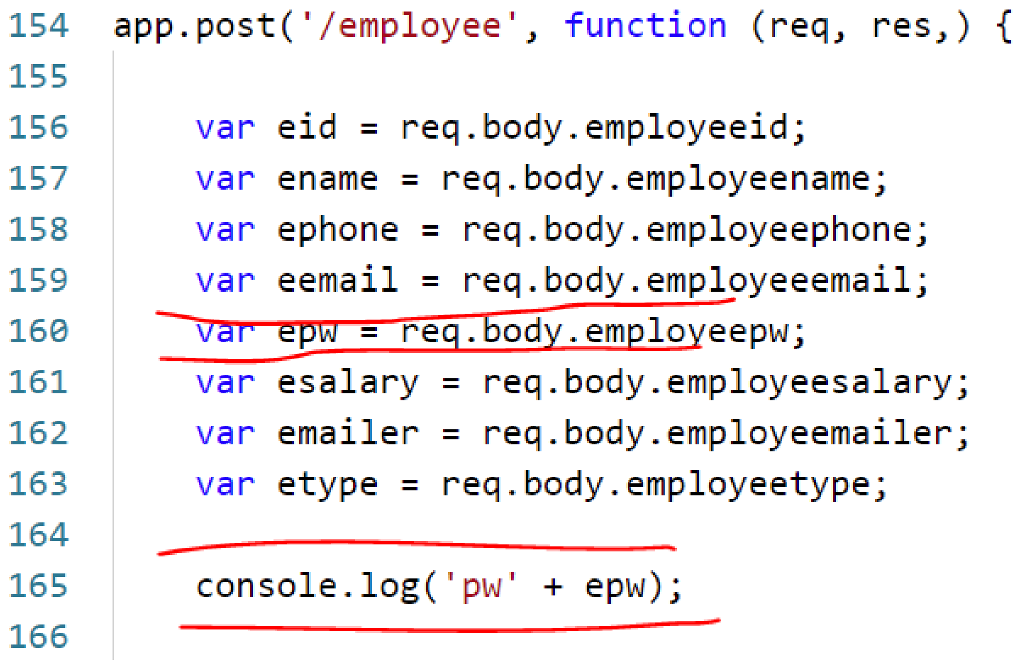
```
} else {
    return (
        <div className={this.props.uniqueName}>
            <input
                type={this.props.inputType}
                name={this.props.uniqueName}
                id={this.props.uniqueName}
                placeholder={this.props.text}
```

From here, save the insertemployees.js page and open server.js. You will need to add the line of code below to add the requirement for the bcrypt libraries to be utilized in the program.



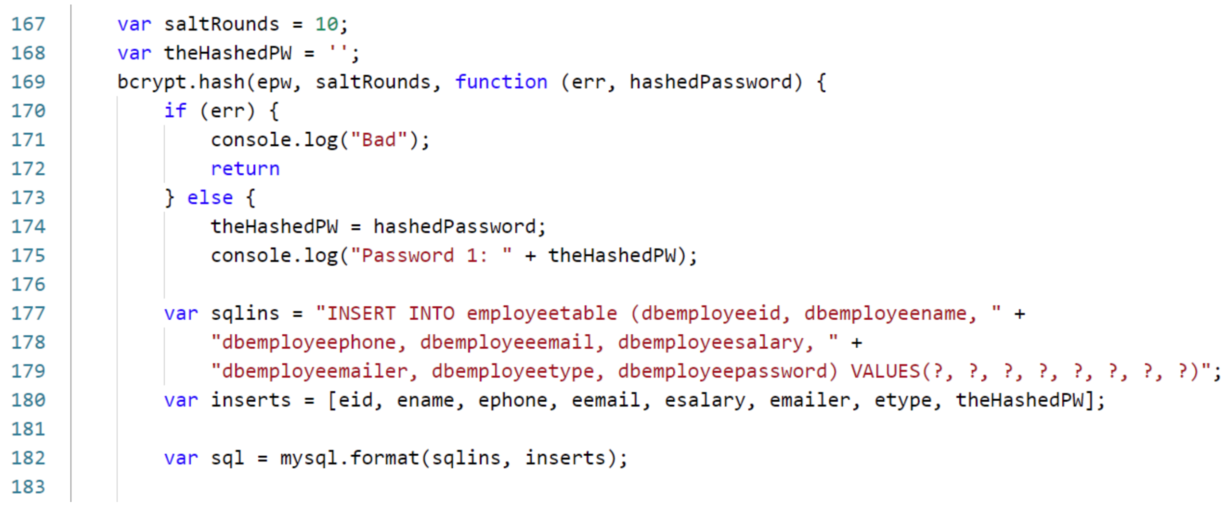
```
var app = express();
var bcrypt = require('bcrypt');

```

Once that is complete, we need to modify the code in the /employee function to be able to save the employee password to the database as an encrypted password. Add the lines of code below to add a variable for the password, as well as output it to the console for debugging purposes.



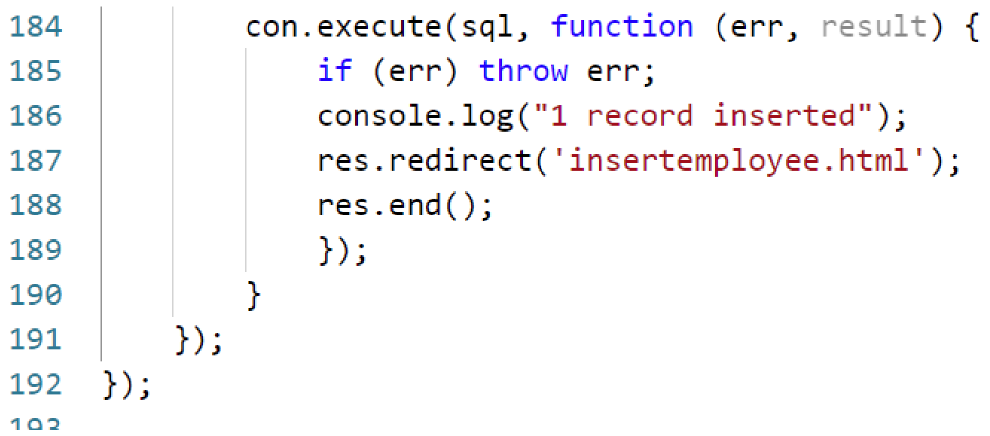
```
app.post('/employee', function (req, res,) {

    var eid = req.body.employeeid;
    var ename = req.body.employeename;
    var ephone = req.body.employeephone;
    var eemail = req.body.employeeemail;
    var epw = req.body.employeepw;
    var esalary = req.body.employeesalary;
    var emailer = req.body.employeemailer;
    var etype = req.body.employeetype;

    console.log('pw' + epw);

```

The code listed below will determine how many rounds to salt the hashed password on line 167, line 168 creates a variable for the hashed password. Line 169 will create the method to hash and salt the password using the bcrypt function. It takes 2 variables, that password to be used and the rounds to salt. It will then either give an error or the hashedPassword. Line 179 creates an error message in case the hashed password creation fails. Line 173 is the else statement for when the creation is successful, all previous code for SQL and running the query will go inside of this if statement. Line 174 sets the theHashedPW and Line 175 outputs it to the console for debugging. Line 177 creates the SQL statement with the password database field and one ? added. Line 180 adds theHashedPW to the array. Line 182 formats the SQL statement for execution.



```
167  var saltRounds = 10;
168  var theHashedPW = '';
169  bcrypt.hash(epw, saltRounds, function (err, hashedPassword) {
170      if (err) {
171          console.log("Bad");
172          return
173      } else {
174          theHashedPW = hashedPassword;
175          console.log("Password 1: " + theHashedPW);
176
177      var sqlins = "INSERT INTO employeetable (dbemployeeid, dbemployeename, " +
178          "dbemployeephone, dbemployeeemail, dbemployeesalary, " +
179          "dbemployeemailer, dbemployeetype, dbemployeepassword) VALUES(?, ?, ?, ?, ?, ?, ?, ?)";
180      var inserts = [eid, ename, ephone, eemail, esalary, emailer, etype, theHashedPW];
181
182      var sql = mysql.format(sqlins, inserts);
183
```

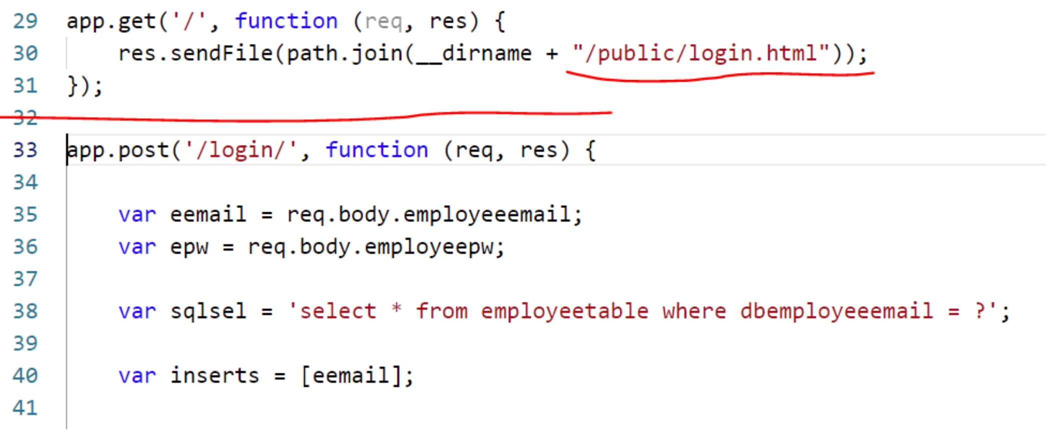
The code below completes the execution, the main thing to review here is to ensure that the brackets and parentheses and semicolons match with your code.



```
184          con.execute(sql, function (err, result) {
185              if (err) throw err;
186              console.log("1 record inserted");
187              res.redirect('insertemployee.html');
188              res.end();
189              });
190          }
191      });
192  });
193
```

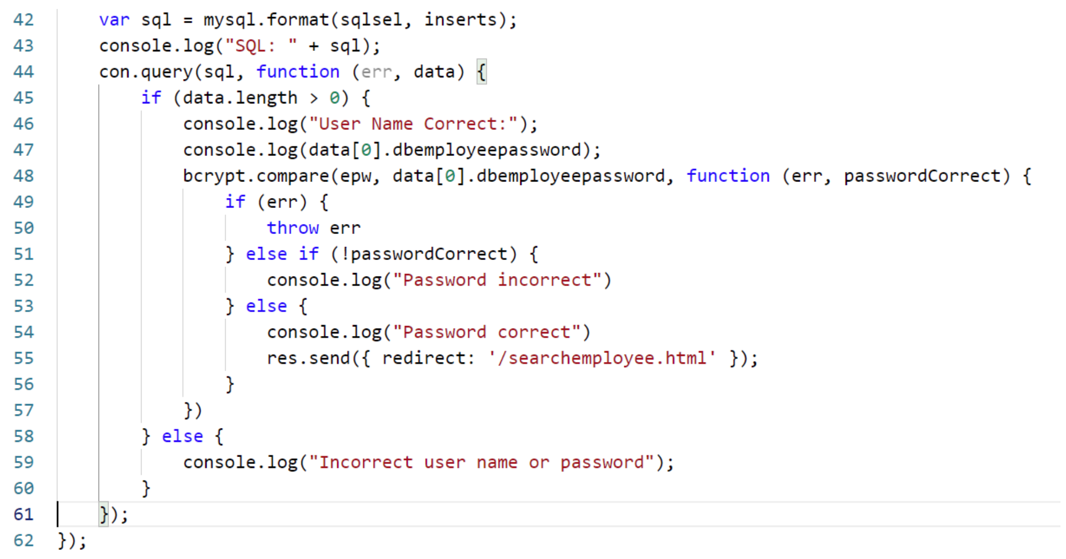
Once this is complete, you will need to run the program and add a user with a password. The user needs to be created with an encrypted password, otherwise you will not be able to login once the login page is created. Make a user is the employee table with a user name of test@test.com and a password of password.

Now that the insertion of the password is complete, we will look at the login pages. The login html and js pages have been provided, the only thing that needs to be added is the login script inside the server.js page. When someone logs in, the information for a user needs to be selected from the database and compared to the information given by the user when they login. Add the code below at the top of the server.js, below where the app.get that opens the starting page resides. Change the code below to switch the starting page to login.html. The code below creates a function named /login/ that takes an email and pw as variables on lines 35 and 36. A SQL statement is then made that selects all records from the employeetable where the email equals whatever was given on the login form. The variable from the login form is placed into the array on line 40.



```
29  app.get('/', function (req, res) {
30      res.sendFile(path.join(__dirname + "/public/login.html"));
31  });
32
33  app.post('/login/', function (req, res) {
34
35      var eemail = req.body.employeeemail;
36      var epw = req.body.employeepw;
37
38      var sqlsel = 'select * from employeetable where dbemployeeemail = ?';
39
40      var inserts = [eemail];
41
```

Line 42 below formats the SQL statement and it is output for debugging on line 43. A query is then run on line 44, line 45 checks to see if there are returned results from the query, which means the User name matches one in the database. Line 47 grabs the password from the data array which is the result set from the database query. Line 48 is the bcrypt command to compare a password from the database to the password given in the form. The password in the form is hashed and then compared so that the hashed password is never decrypted. There are several if statements, one for if there is an error, one for if the passwords do not match and the last one will show the password was correct, and then redirect the user to the searchemployee page.



```
42      var sql = mysql.format(sqlsel, inserts);
43      console.log("SQL: " + sql);
44      con.query(sql, function (err, data) {
45          if (data.length > 0) {
46              console.log("User Name Correct:");
47              console.log(data[0].dbemployeepassword);
48              bcrypt.compare(epw, data[0].dbemployeepassword, function (err, passwordCorrect) {
49                  if (err) {
50                      throw err
51                  } else if (!passwordCorrect) {
52                      console.log("Password incorrect")
53                  } else {
54                      console.log("Password correct")
55                      res.send({ redirect: '/searchemployee.html' });
56                  }
57              })
58          } else {
59              console.log("Incorrect user name or password");
60          }
61      });
62  });
```

The code given will now allow someone to login to the site. For this rest of this lab, you will need to perform a similar task for the customer table, a password must be added to that table and code make to allow the password to be added through the website. There is nothing that should be done at this time for customer login, just get the ability for a customer to have an encrypted password into the system. In addition, create a customer in the system with an email of user@user.com and a password of password. Once complete, zip and upload to the dropbox in D2L.