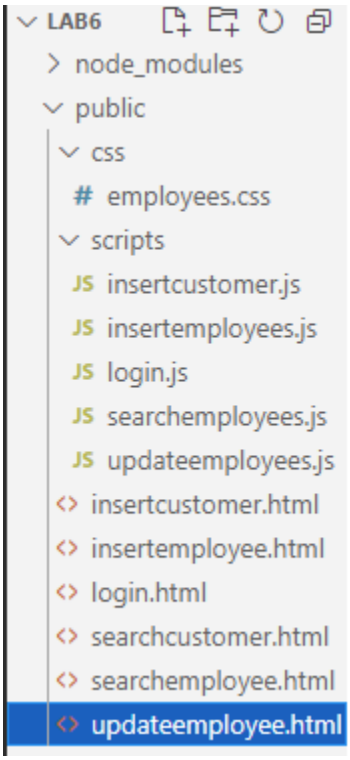
Lab 6

For this lab we will be adding a page for updating the employee information. To start, copy all the files from Lab 5 into a folder called lastname_Lab6. Once the files are copied, open the lastname_lab6 folder in VS Code. Go into the package.json file and change the lab5 to lab6. Also, be sure to go into all the html pages and change the title from lab5 to lab6. In addition, the dropbox for D2L has a file for the css for this project, download that file and create a css folder inside the public folder. Copy the css file into the css folder. In addition, the search pages for employee will be the basis for the update pages, so the searchemployee.html page will need to be copied and the new file renamed updateemlpoyee.html. Also, in the scripts folder, the searchemployees.js file will need to be copied and the new file named updateemployees.js. Your folder structure should look as below.



Next, you will need to adapt all of the html pages to access the css file that has been copied. The code below the red line will add the link to the css file. The css will just add borders to the tables and allow formatting to occur with where divs will be placed on our form. You will also need to add a link to the updateemployee page on all html pages as in the image below.

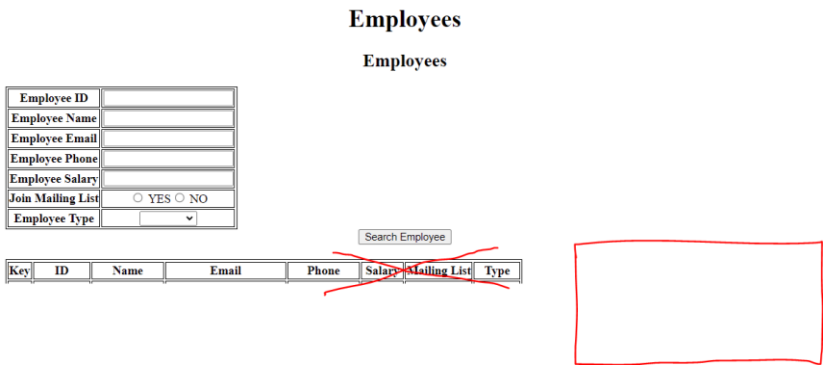
```
5     <title>Lab 6</title>
6     <link rel="stylesheet" href="css/employees.css" />
```

```
19        <a href="searchemployee.html">Search Employee</a>
20        <a href="updateemployee.html">Update Employee</a>
```

Open the newly created updateemployee.html page. Update the link to the js page on this html page to link to updateemployees.js as below.

```
22    <div id="content"></div>
23    <script type="text/babel" src="scripts/updateemployees.js">
24    </script>
```

The update employees page should look like the search page, with a form at the top and a list of results at the bottom. In the image below, the box in red is where the area where information will be updated will be shown. Also, we do not need to output all the information to allow an employee to be updated, so the fields crossed out below will be removed from the list output. We will still be able to search by all fields, and a new button will be added to the list that allows up to hit update and populate the new form that will be created on the right.



To do this, open the updateepmployee.js page. We will start by adding a function to update a single employee from the server. This will create a command similar to the command to send information to the server when something is searched. However, this code will send the data over to the server.js page and look for the function named /updatesingleemp, which will be created later. Add the code between the red lines below.

```
37    },
38    updateSingleEmpFromServer: function (employee) {
39
40        $.ajax({
41            url: '/updatesingleemp',
42            dataType: 'json',
43            data: employee,
44            type: 'POST',
45            cache: false,
46            success: function (upsingledata) {
47                this.setState({ upsingledata: upsingledata });
48            }.bind(this),
49            error: function (xhr, status, err) {
50                console.error(this.props.url, status, err.toString());
51            }.bind(this)
52        });
53        window.location.reload(true);
54    },
55    componentDidMount: function () {
```

Now the page will need to be formatted to handle the additional form that will be on the screen. Add line 63 below, which will now show update on the page. Lines 66 and 67 will create the divs that will allow the ability to move the forms left or right on the page.

```
63          <h1>Update Employees</h1>
64          <Employeeform2 onEmployeeSubmit={this.loadEmployeesFromServer} />
65          <br />
66          <div id = "theresults">
67              <div id = "theleft">
68              <table>
```

The output that will show will be limited from the search page, as we do not need to see all the information on the update page. The lines in red below on the left will need to be removed, and the one starting and closing <th> will need to be added. The image on the right is what is should look like when the changes are complete.

```
<tr>
    <th>Key</th>
    <th>ID</th>
    <th>Name</th>
    <th>Email</th>
    <th>Phone</th>
    <th>Salary</th>
    <th>Mailing List</th>
    <th>Type</th>
</tr>
```

```
70          <tr>
71              <th>Key</th>
72              <th>ID</th>
73              <th>Name</th>
74              <th>Email</th>
75              <th></th>
76          </tr>
77      </thead>
```

The form that will need to be created on the right side of the screen will be added in the code between the red lines below. This code closes several of the divs that were created above, while also creating a new div that will end up on the right side of the screen. Line 82 will call the EmployeeUpdateForm class that will be created alter and instantiate that object.

```
79              </table>
80              </div>
81              <div id="theright">
82                  <EmployeeUpdateform onUpdateSubmit={this.updateSingleEmpFromServer} />
83              </div>
84          </div>
85      </div>
```

Since we will need the primary key to update our record, add a new variable for a key field as shown on Line 93 below.

```
92      return {
93          employeekey: "",
94          employeeid: "",
```

In the form previously created for the search area, several div tags will be created to better format these forms on the page. The lines in red below will add the divs for the form.

```
154         return (
155         <div>
156             <div id = "theform">
157             <form onSubmit={this.handleSubmit}>
```

The code below will close some of the previously created divs, while also creating a new form with a submit button that will clear the search form.

```
222             </form>
223             </div>
224             <div>
225                     <br />
226                     <form onSubmit={this.getInitialState}>
227                         <input type="submit" value="Clear Form" />
228                     </form>
229             </div>
230         </div>
231         );
```

The next code to the created will be for the form that needs to be created for the update area. This will be similar to the form that allows the user to enter search information. All the variables that the form will handle will need to be created, as well as the class as shown in the code below.

```
234
235 var EmployeeUpdateform = React.createClass({
236     getInitialState: function () {
237         return {
238             upemployeekey: "",
239             upemployeeid: "",
240             upemployeename: "",
241             upemployeeemail: "",
242             upemployeephone: "",
243             upemployeesalary: "",
244             upemployeeMailer: "",
245             upselectedOption: "",
246             updata: []
247         };
248     },
```

The code below creates the function to update options when they change on lines 249 to 253. Lines 254 to 266 create the code to access the /getemptypes method from server.js which populates the data from the table for the drop down list of employee types. Lines 267 to 270 will run the code to load the employee types.

```
249     handleUpOptionChange: function (e) {
250         this.setState({
251             upselectedOption: e.target.value
252         });
253     },
254     loadEmpTypes: function () {
255         $.ajax({
256             url: '/getemptypes',
257             dataType: 'json',
258             cache: false,
259             success: function (data) {
260                 this.setState({ updata: data });
261             }.bind(this),
262             error: function (xhr, status, err) {
263                 console.error(this.props.url, status, err.toString());
264             }.bind(this)
265         });
266     },
267     componentDidMount: function () {
268         this.loadEmpTypes();
269 
270     },
```

A function to handle the data to be passed to the server page will need to be added for when the submit button is clicked. The code below creates the function, stops the default click event from happening, and creates all the variables that will come from the form.

```
271     handleUpSubmit: function (e) {
272         e.preventDefault();
273 
274         var upemployeekey = upempkey.value;
275         var upemployeeid = upempid.value;
276         var upemployeeemail = upempemail.value;
277         var upemployeename = upempname.value;
278         var upemployeephone = upempphone.value;
279         var upemployeesalary = upempsalary.value;
280         var upemployeemailer = this.state.upselectedOption;
281         var upemployeetype = upemptype.value;
```

The code below updates the form variables to prepare them to be passed to the server.js page. Then the function is created for when a change is made on the form, similar to the search form above.

```
282
283         this.props.onUpdateSubmit({
284             upemployeekey: upemployeekey,
285             upemployeeid: upemployeeid,
286             upemployeename: upemployeename,
287             upemployeeemail: upemployeeemail,
288             upemployeephone: upemployeephone,
289             upemployeesalary: upemployeesalary,
290             upemployeemailer: upemployeemailer,
291             upemployeetype: upemployeetype
292         });
293     },
294     handleUpChange: function (event) {
295         this.setState({
296             [event.target.id]: event.target.value
297         });
298     },
```

The actual form will now need to be rendered so it can be viewed on the screen. The code below creates the render function and sets the return to the html code. Two divs will be created as well as a form. A starting table and tbody tag are also created.

```
299     render: function () {
300
301         return (
302             <div>
303                 <div id="theform">
304                     <form onSubmit={this.handleUpSubmit}>
305
306                         <table>
307                             <tbody>
```

The table rows and data tags for each of the fields will need to be created, as well as the input tags. The code below is very similar to the tags created for the search fields, the main difference will be the variable names for the input fields will be different since this is a different form. The code below is the first 3 fields on the form.

```
    <tr>
        <th>Employee ID</th>
        <td>
<input type="text" name="upempid" id="upempid" value={this.state.upempid} onChange={this.handleUpChange} />
        </td>
    </tr>
    <tr>
        <th>Employee Name</th>
        <td>
<input name="upempname" id="upempname" value={this.state.upempname} onChange={this.handleUpChange} />
        </td>
    </tr>
    <tr>
        <th>Employee Email</th>
        <td>
<input name="upempemail" id="upempemail" value={this.state.upempemail} onChange={this.handleUpChange} />
        </td>
    </tr>
```

The code below will be for the next 2 fields, phone and salary.

```
    <tr>
        <th>Employee Phone</th>
        <td>
<input name="upempphone" id="upempphone" value={this.state.upempphone} onChange={this.handleUpChange} />
        </td>
    </tr>
    <tr>
        <th>Employee Salary</th>
        <td>
<input name="upempsalary" id="upempsalary" value={this.state.upempsalary} onChange={this.handleUpChange} />
        </td>
    </tr>
```

The mailing list code below is slightly different, since this will be a radio button. This is just for the one button for YES.

```
    <tr>
        <th>
            Join Mailing List
        </th>
        <td>
            <input
                type="radio"
                name="upempmailer"
                id="upempmaileryes"
                value="1"
                checked={this.state.upselectedOption === "1"}
                onChange={this.handleUpOptionChange}
                className="form-check-input"
            />Yes
```

This code creates the other radio button for NO.

```
352                 <input
353                 type="radio"
354                 name="upempmailer"
355                 id="upempmailerno"
356                 value="0"
357                 checked={this.state.upselectedOption === "0"}
358                 onChange={this.handleUpOptionChange}
359                 className="form-check-input"
360             />No
361         </td>
362     </tr>
```

The code below creates the table row and data for the employee type drop down, as well as closed the table body and table tags. Line 373 created a hidden field so that the employee key can be passed to the server.js page so the system knows which record to update. Line 374 creates the submit button. The rest of the code is closing tags.

```
363     <tr>
364         <th>
365             Employee Type
366         </th>
367         <td>
368             <SelectUpdateList data={this.state.updata} />
369         </td>
370     </tr>
371 </tbody>
372                     </table><br />
373                     <input type="hidden" name="upempkey" id="upempkey" onChange={this.handleUpChange} />
374                     <input type="submit" value="Update Employee" />
375                 </form>
376             </div>
377         </div>
378         );
379     }
380 });
381
```

Since some of the fields were removed from the table that shows the results of the search, these fields will need to be removed from the EmployeeList class as well. Remove the 4 fields from the left image, the final result should look like the image on the right.

```
return (
    <Employee
        key={employee.dbemployeekey}
        empkey={employee.dbemployeekey}
        empid={employee.dbemployeeid}
        empname={employee.dbemployeename}
        empemail={employee.dbemployeeemail}
        empphone={employee.dbemployeephone}
        empsalary={employee.dbemployeesalary}
        empmailer={employee.dbemployeemailer}
        emptype={employee.dbemptypename}
    >
    </Employee>
);
```

```
385     return (
386         <Employee
387             key={employee.dbemployeekey}
388             empkey={employee.dbemployeekey}
389             empid={employee.dbemployeeid}
390             empname={employee.dbemployeename}
391             empemail={employee.dbemployeeemail}
392         >
393         </Employee>
394     );
```

Now that the outputs for the list are updated, the code to grab information for a single employee and output to the form on the right when a button is clicked will be created. The Employee class will be updated to add the code below the red line. The initial state is set with the employee key value so the program has that for the update command later, as well as the result set of the employee information grabbed from the server. An updateRecord function is created for the button click, which will create a variable for the employee key, as well as run a function to load a single employee using the employee key.

```
407 var Employee = React.createClass({
408     getInitialState: function () {
409         return {
410             upempkey: "",
411             singledata: []
412         };
413     },
414     updateRecord: function (e) {
415         e.preventDefault();
416         var theupempkey = this.props.empkey;
417 
418         this.loadSingleEmp(theupempkey);
419     },
```

The code below starts to create the loadSingleEmp function, having one parameter, which will be the employee key. The url will link to a function on the server.js page, with the data being used the employee key. This function is a bit different from previous functions as it will both receive and send data to the server.js page.

```
420     loadSingleEmp: function (theupempkey) {
421         $.ajax({
422             url: '/getsingleemp',
423             data: {
424                 'upempkey': theupempkey
425             },
426             dataType: 'json',
427             cache: false,
```

The code below is the success function for the function above. The state of the singledata array is populated with the data returned, and a variable is then created called populateEmp. This variable will hold all the information from the database about the single employee, with lines 432 to 443 updating the right side form fields to have the values given from the database.

```
428             success: function (data) {
429                 this.setState({ singledata: data });
430                 console.log(this.state.singledata);
431                 var populateEmp = this.state.singledata.map(function (employee) {
432                     upempkey.value = theupempkey;
433                     upempemail.value = employee.dbemployeeemail;
434                     upempid.value = employee.dbemployeeid;
435                     upempphone.value = employee.dbemployeephone;
436                     upempsalary.value = employee.dbemployeesalary;
437                     upempname.value = employee.dbemployeename;
438                     if (employee.dbemployeemailer == 1) {
439                         upempmaileryes.checked = true;
440                     } else {
441                         upempmailerno.checked = true;
442                     }
443                     upemptype.value = employee.dbemployeetype;
444 
445                 });
446             }.bind(this),
```

The code below adds the error function, so if the data does not get received from the server.js page correctly, an error is thrown.

```
447             error: function (xhr, status, err) {
448                 console.error(this.props.url, status, err.toString());
449             }.bind(this)
450         });
451 
452     },
453 
```

The next portion of the Employee class deals with outputting the Employee information to the list on the left, so the fields that we removed will need to be removed from here as well. These fields are crossed out on the left below. On the right is the new code that will place a button to update the employee information.

```
<td>
    {this.props.empemail}
</td>
<td>
    {this.props.empphone}
</td>
<td>
    {this.props.empsalary}
</td>
<td>
    {themailer}
</td>
<td>
    {this.props.emptype}
</td>
```

```
474                 <td>
475                     {this.props.empemail}
476                 </td>
477                 <td>
478                     <form onSubmit={this.updateRecord}>
479                         <input type="submit" value="Update Record" />
480                     </form>
481                 </td>
482             </tr>
```

The final item for this page is a copy of the SelectList function that is used on the search form. This copy is for the form on the right when we update, since we cannot reuse the code for both forms. The main difference in the 2 Classes are underlined in red below, mainly just variable names for the fields.

```
507
508 var SelectUpdateList = React.createClass({
509     render: function () {
510         var optionNodes = this.props.data.map(function (empTypes) {
511             return (
512                 <option
513                     key={empTypes.dbemptypeid}
514                     value={empTypes.dbemptypeid}
515                 >
516                     {empTypes.dbemptypename}
517                 </option>
518             );
519         });
520         return (
521             <select name="upemptype" id="upemptype">
522                 <option value="0"></option>
523                 {optionNodes}
524             </select>
525         );
526     }
527 });
528
```

Now that the updateemployee.js page is complete, there are 2 functions that need to be created on the server.js page. One is to get the single employee information, the other is to update the single employee. The get will be the first to be created below. This can be placed at any place in server.js between other functions. This works the same as the previous select statements we have made for searches, taking one value, then running a select statement and placing the variable into an array. Then the sql statement is formatted and the query is run.

```
79
80  app.get('/getsingleemp/', function (req, res) {
81
82      var ekey = req.query.upempkey;
83
84      var sqlsel = 'select * from employeetable where dbemployeekey = ?';
85      var inserts = [ekey];
86
87      var sql = mysql.format(sqlsel, inserts);
88
89      con.query(sql, function (err, data) {
90          if (err) {
91              console.error(err);
92              process.exit(1);
93          }
94
95          res.send(JSON.stringify(data));
96      });
97  });
98
```

The next item is a function to update the employee, note that this function is an app.post. After the creation, variables are created for all the form fields.

```
99  app.post('/updatesingleemp', function (req, res, ) {
100
101     var eid = req.body.upemployeeid;
102     var ename = req.body.upemployeename;
103     var ephone = req.body.upemployeephone;
104     var eemail = req.body.upemployeeemail;
105     var esalary = req.body.upemployeesalary;
106     var emailer = req.body.upemployeemailer;
107     var etype = req.body.upemployeetype;
108     var ekey = req.body.upemployeekey;
109
```

The code below creates the update statement in SQL, then an array for the variables is created, followed by the command to format the SQL statement. The statement is then executed to update the information on the server.

```
110     var sqlins = "UPDATE employeetable SET dbemployeeid = ?, dbemployeename = ?, dbemployeeemail = ?, " +
111         " dbemployeephone = ?, dbemployeesalary = ?, dbemployeemailer = ?, dbemployeetype =? " +
112         " WHERE dbemployeekey = ? ";
113     var inserts = [eid, ename, eemail, ephone, esalary, emailer, etype, ekey];
114
115     var sql = mysql.format(sqlins, inserts);
116 console.log(sql);
117     con.execute(sql, function (err, result) {
118             if (err) throw err;
119             console.log("1 record updated");
120
121             res.end();
122         });
123 });
124
```

This will complete the code to update employee information. For this rest of this lab, you will need to perform a similar task for the customer table, a page must be created to allow the user to select a customer and update their information. Once complete, zip and upload to the dropbox in D2L.