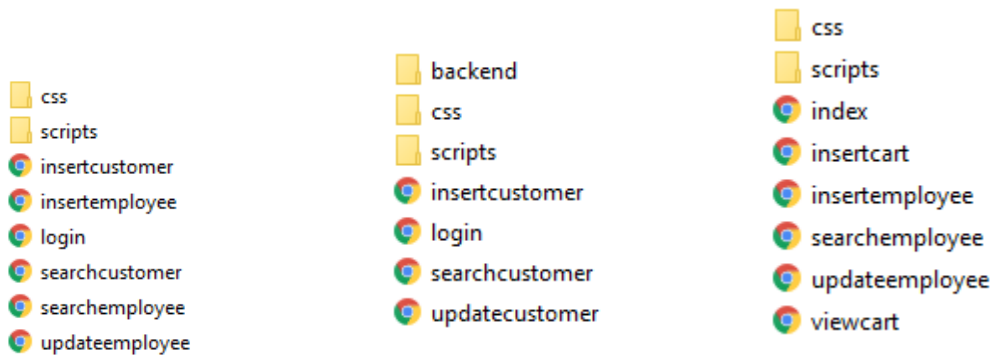


Lab 7

For this lab we will be adding a page for updating the employee information. To start, copy all the files from Lab 6 into a folder called lastname_Lab7. Once the files are copied, open the lastname_lab7 folder in VS Code. Go into the package.json file and change the lab6 to lab7. Also, be sure to go into all the html pages and change the title from lab6 to lab7. This lab will be modifying the structure for the previous lab to handle both a front end and back end.

To begin, a folder names backend will be placed into the public folder. From there, the css and scripts folders, as well as the login.html page will be COPIED into the backend folder. For this example, the employee pages will be on the backend and the customer pages will be on the frontend. So, all the employee pages should be MOVED into the backend folder. In the scripts folder on the front end, the employee js files can be removed. In the scripts folder on the backend, the customer js files can be removed. Inside the backend folder, change the name of the login.html to index.html. Also, the dropbox for this assignment has 4 files, ones for viewcart.html and insertcart.html and ones for viewcarts.js and insertcart.js. Copy the html files into the backend folder, and the js files into the scripts folder on the backend. The images below show the starting folder structure on the left, the public folder in the middle and the backend folder on the right.



At this point the program has 2 login pages, one on the front end called login, and one on the backend called index. Both have the same js file called login. Though it will function this way, some changes will need to be made to ensure the pages are unique for front and back end. The login.js page on the backend will need to be changed to loginemp.js, since it is only for the employee. Once complete, the js files in the scripts folder in the backend folder should look like the left image below. The link to the script on the index.html page will also need to be changed to link to the renamed file for the loginemp.js as below on the right image.



Since there will eventually be 2 login pages that could access the server.js page, the code for the login that was created for employees will need to be changed to specifically work for just logging in employees. This can be easily achieved by changing just the name of the function that was created for login. The other change that would need to be made is the link to the page that will be redirected to after login will need the correct path to the backend. Below, the /login/ function has been changed to utilize /loginemp/.

```
32
33 app.post('/loginemp/', function (req, res) {
34     var eemail = req.body.employeeemail;
```

The other item to be changed in this function is the redirect when the login occurs. Since the `server.js` is in the root directory of the program, and the `searchemployee.html` page is in the backend folder now, the full path to the file must be given.

```
56 console.log("Password Correct");
57 res.send({ redirect:  '/backend/searchemployee.html' });
58 }
59 ...

```


Now open up the loginemp.js file, the links to the previous /login/ function there will need to be changed to /loginemp/ as in the image below.

```
9      $.ajax({
10          url: '/loginemp/',
11          dataType: 'json',
```

This will complete the code to make the new login area for the backend function. Next, the links to the new insert and view cart pages should be added to ALL the html pages, but only on the backend. The image below shows the links that will be required on the backend, along with some separator characters to make the nav area a bit more aesthetically pleasing.

```
16 <nav>
17     <a href="insertemployee.html">Insert Employee</a> |
18     <a href="insertcart.html">Insert Cart</a> |
19     <a href="searchemployee.html">Search Employee</a> |
20     <a href="updateemployee.html">Update Employee</a> |
21     <a href="viewcart.html">View Carts</a>
22 </nav>
```

The purpose of the cart pages is to be able to insert and view carts that can be added for purchasing reasons. The cart the will be created will not have any items in it at this time. It will be created as a holder for the basic information for a cart, such as the employee and customer who are linked to the cart, the date the cart was created, the daily id number of the cart and whether or not the cart has been made or picked up. The dbcartid is a primary key which is auto incremented. The database structure is below, this table should be created in the database, either manually, or a sql dump file has been provided in the dropbox as well.

<input type="checkbox"/>	1	dbcartid 	int(11)
<input type="checkbox"/>	2	dbcartemp	int(11)
<input type="checkbox"/>	3	dbcartcust	int(11)
<input type="checkbox"/>	4	dbcartdate	datetime
<input type="checkbox"/>	5	dbcartdailyid	int(11)
<input type="checkbox"/>	6	dbcartpickup	int(11)
<input type="checkbox"/>	7	dbcartmade	int(11)

The insert cart page should look like below, all that is there at this point is a drop down for the employee, once an employee is selected, the date will be automatically entered, as well as the daily id will be calculated and added. The pickup and made field will be filled with default values. At this point nothing will be done for the customer field.

[Insert Employee](#) | [Insert Cart](#) | [Search Employee](#) | [Update Employee](#) | [View Carts](#)

Insert Carts

Cart Employee	Jason Carman ▼
----------------------	----------------

Insert Cart

The code on the server.js page to allow this page to function still needs to be created. Two new functions need to be created, one to create the cart, and one to create the drop down that will house all the employees. The function for the employees is going to be named /getemps and the function for the cart will be named /Cart. Note the capitalization for the Cart function, this will be required as the function in the provided insertcart.js will utilize the capitalization.

The code below is for the /getemps function and will need to be added to the server.js page. This will select all the items from the employee table and then all that information to be sent back to be used in the drop down for selecting the employees. This function has the same functionality as the ones created for previous drop downs for employee types.

```
100  app.get('/getemps/', function (req, res) {
101
102      var sqlsel = 'select * from employeetable';
103      var sql = mysql.format(sqlsel);
104
105      con.query(sql, function (err, data) {
106          if (err) {
107              console.error(err);
108              process.exit(1);
109          }
110
111          res.send(JSON.stringify(data));
112      });
113  });
114
```

The other function to be created is the /Cart/ function. The function creation is done below on line 306. This function takes one variable, the employee information given from the dropdown. The variable is created on line 308. The daily id is determined by looking at the current date and then seeing if there are already daily id numbers that exist. If so, the maximum value of the daily id is retrieved. This is done by the code on lines 310-311. The max of the daily id where the date for the cart is the current date is what the code is performing. The command CURDATE() gets the current date. Line 313 formats the sql statement, and line 315 creates a variable for the daily number set to 1. It is set to 1 by default as this is the starting value for daily id numbers over the course of a standard day.

```
305
306  app.post('/Cart/', function (req, res) {
307
308      var cartemp = req.body.CartEmp;
309
310      var sqlsel = 'select MAX(dbcartdailyid) as daymax from cartinfo '
311                  + ' WHERE DATE(dbcartdate) = CURDATE()';
312
313      var sql = mysql.format(sqlsel);
314
315      var dailynumber = 1;
316
```

Line 317 below will start to run the query, with line 318 outputting the results from the query above to see if there is a max value for the daily id for the current day. This is not required, it is there for debugging purposes. The if statements from line 320 – 323 will check to see if there are any results from the query above, and if not ensure that the daily number is set to 1 as it will be the first cart created for the day. If there are results, that means there have already been carts created for the day, it will add one to the max value and create the new cart number.

```
317     con.query(sql, function (err, data) {
318         console.log(data[0].daymax);
319
320         if (!data[0].daymax) {
321             dailynumber = 1;
322         } else {
323             dailynumber = data[0].daymax + 1;
324         }
325     }
```

The insert query is created on lines 326-327. For the date, the command now() is utilized in this instance to get the date and time at the exact moment the data is inserted into the database. Line 328 will create the array to bind variables, and line 329 will format the sql command with the array.

```
326     var sqlinscart = "INSERT INTO cartinfo (dbcartemp, dbcartdailyid, "
327     + "    dbcartpickup, dbcartmade, dbcartdate) VALUES (?, ?, ?, ?, now())";
328     var insertscart = [cartemp, dailynumber, 0, 0];
329
330     var sqlcart = mysql.format(sqlinscart, insertscart);
331
```

The last of the code for this function is below, it will execute the sql command, show a console.log that the insert functioned and redirect the page back to the insertcart.html.

```
332     con.execute(sqlcart, function (err, result) {
333         if (err) throw err;
334         console.log("1 record inserted");
335         res.redirect('insertcart.html');
336         res.end();
337     });
338 });
339 });
```

This will complete the code for inserting a cart. The next item to enter on the server.js is the code for the view cart page. The view cart page should look as below, It is a page that allows the user to select an employee and then see all carts that were created by that employee.

[Insert Employee](#) | [Insert Cart](#) | [Search Employee](#) | [Update Employee](#) | [View Carts](#)

View Carts

Cart Employee	19 - Jason Carman ▼
---------------	---------------------

View Cart

ID	Employee	Date	Daily ID	Pickup	Made
----	----------	------	----------	--------	------

The view carts page has a drop down for the employee, just the way the insert cart page did. However, we do not have to create another function on the server.js page for this drop down. The /getemps/ function created previously will be reutilized for this. The code below shows the /getcart/ function that will make this page work. The employee id is created as a variable on line 343. The sql command on lines 345 – 347 will select all the information from the cartinfo table, and join that information with the employee name from the employee table. This will only occur for employees that have the employee id selected from the drop down. The rest of the code formats the sql command, and runs the query. The data returned is then sent back to the view cart page.

```

340
341 app.get('/getcart/', function (req, res) {
342
343     var empid = req.query.employeeid;
344
345     var sqlsel = 'Select cartinfo.*, employeetable.dbemployeename from cartinfo' +
346         ' inner join employeetable on employeetable.dbemployeekey = cartinfo.dbcartemp' +
347         ' where dbcartemp = ? ';
348
349     var inserts = [empid];
350
351     var sql = mysql.format(sqlsel, inserts);
352
353     console.log(sql);
354
355     con.query(sql, function (err, data) {
356         if (err) {
357             console.error(err);
358             process.exit(1);
359         }
360         res.send(JSON.stringify(data));
361     });
362 });

```

This will complete the code for the backend. For this rest of this lab, you will need to perform a similar task for login on the front end, the login page on the front end must allow the customer to login and the js file should be renamed logincust.js. Remember, the passwords should be encrypted and you will need to create users with the appropriate credentials for login. The frontend should have a customer with an email of user@user.com and a password of password. The backend should have an employee with an email of test@test.com and a password of password. Once complete, zip and upload to the dropbox in D2L.