REACT EXPRESS AND MYSQL

BY QCC - TECH WORKS

GOAL

- Use our React and Express boilerplate
- Connect and do CRUD operations on a mySQL DB

WE HAVE AN IDEA!! BUT WE HAVE A BOILERPLATE!!! AHH YEAH

- What do we do first when we have an idea for an application??? We create a directory that we're going to place this application in.
- You still want to make a new repo for this project, we can call it reactexpressandmysql.
- Then you want to clone your react and express boilerplate into your new directory.

- Since we cloned our repo into our projects directory. We're going to have to install the dependencies that are already defined in our package.json file. Lets install them by running the command "npm install" in the command prompt. Be sure that you're in the projects directory! Then just to make sure it got all the packages... we need to change directories into our CLIENT folder and run the same command to get all the dependencies for our react app as well.

- Now we need to bring in the mySQL dependency. In the root directory of your project. You want to run the command “npm install mysql”
- Okay! We have all our dependencies installed lets start coding.

- What we need to do is first navigate to our server.js file. Lets bring in our mysql package, by adding this line of code to the top of the module.

```
1  const mysql = require('mysql');
```

- Then we have to make our connection to our DB. We're going to try to connect to the QCC techworks DB.

```
6
7   const mysqlConnection = mysql.createConnection({
8     host: '10.9.3.218',
9     user: 'TWStudent',
.0    password: 'TechWorks!',
.1    database: 'employeedb',
.2    multipleStatements: true
.3  });
```

- Lets set this up where we get this connection running and see if we're getting errors or the connection was successful. To do this we want to add these lines of code:

```
14  mysqlConnection.connect(err => {
15    if (!err) console.log(`DB connection succeded`);
16    else {
17      console.log(
18        `DB connection failed Error: ` + JSON.stringify(err, undefined, 2)
19      );
20    }
21  });
```

- Now to test this we should go to our console and make sure we are in the parent directory of our project and type in “node server.js” I’m doing this just to test the connection… I don’t need the whole dev environment up and running i just want to test the connection before i spin everything else up.

- If successful you should see the console log of “DB connection successful” or you should see the error if we are unsuccessful.

- Now that the DB is connected we can start doing using queries to produce data that we can use. Lets first start by adding a get route that will grab all our employees.

```
32  //Get all employees
33  app.get('/employees', (req, res) => {
34      mysqlConnection.query('SELECT * FROM Employee', (err, rows, field) => {
35          if (!err) console.log(rows);
36          else console.log(err);
37      });
38  });
```

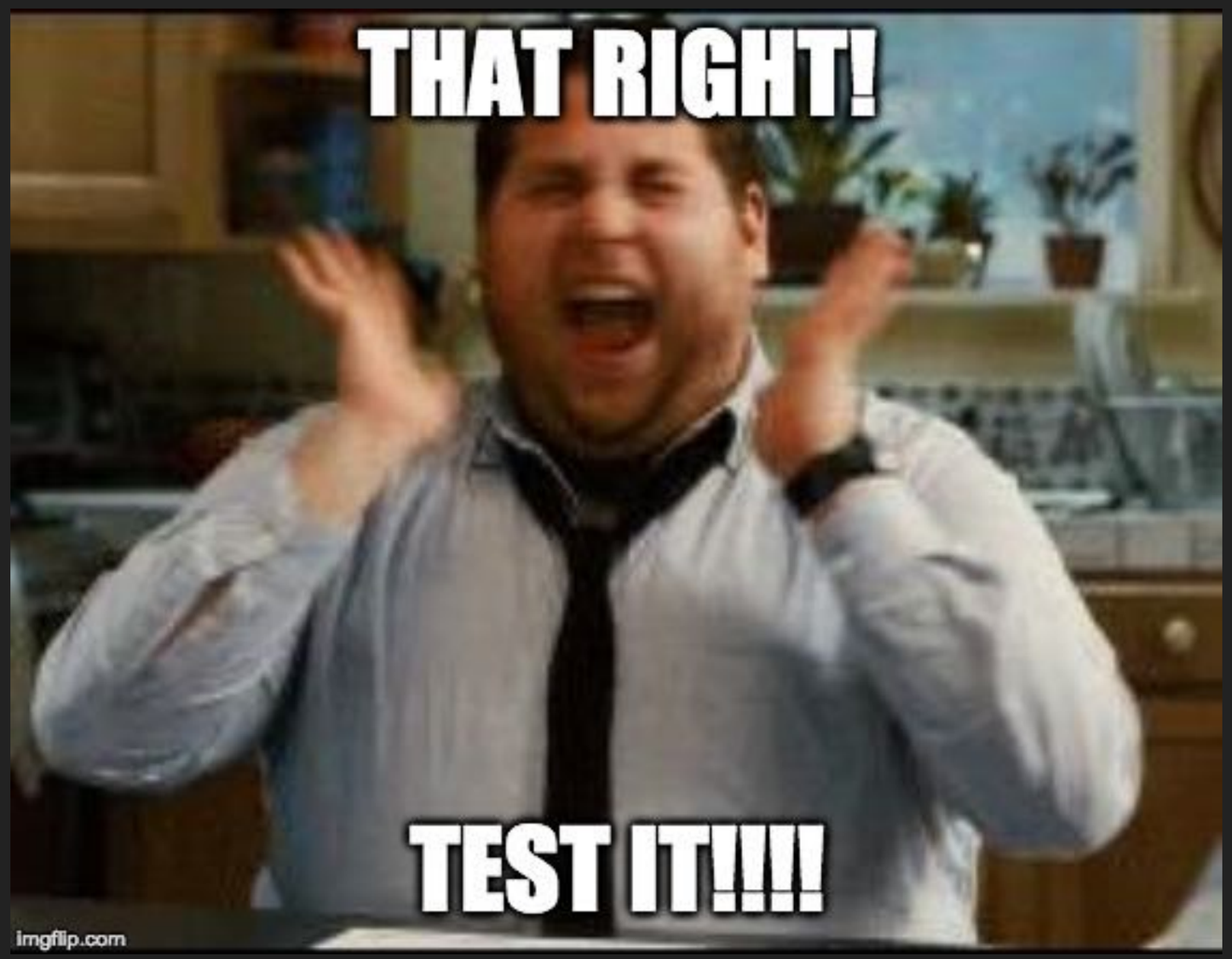
- Now that we have a route set up how can we test this??

- You can test this by opening your browser and navigating to your http://localhost:4000/employee
- Or you could use Postman to test the route and see if you get the correct data back.

- Now that we have our first get route working correctly we need to build another route that will allow us to specifically get the single employee we're searching for. We can accomplish this by adding "/:id" to our employees route and then changing the query to be specific to the parameters we're specifying (in this case its the id) and then send back the rows.

```
41  //Get an employee
42  app.get('/employees/:id', (req, res) => {
43    mysqlConnection.query(
44      'SELECT * FROM Employee WHERE EmpID =?',
45      [req.params.id],
46      (err, rows, field) => {
47        if (!err) res.send(rows);
48        else console.log(err);
49      }
50    );
51  });
```

- Now that we have a route set up test it! - In your browser or in postman.



- Let's keep it hot and add a delete route. This route is a lot like the get route for obtaining the employee by the specific id... We have to change the route to app.delete and we have to change the query as well. This query is simply Deleting the employee by the ID. We also want to change the response we send to the user by just adding some text.

```
53  //Delete an employee
54  app.delete('/employees/:id', (req, res) => {
55      mysqlConnection.query(
56          'DELETE FROM Employee WHERE EmpID =?',
57          [req.params.id],
58          (err, rows, field) => {
59              if (!err) res.send('Deleted Successfully');
60              else console.log(err);
61          }
62      );
63  });
```

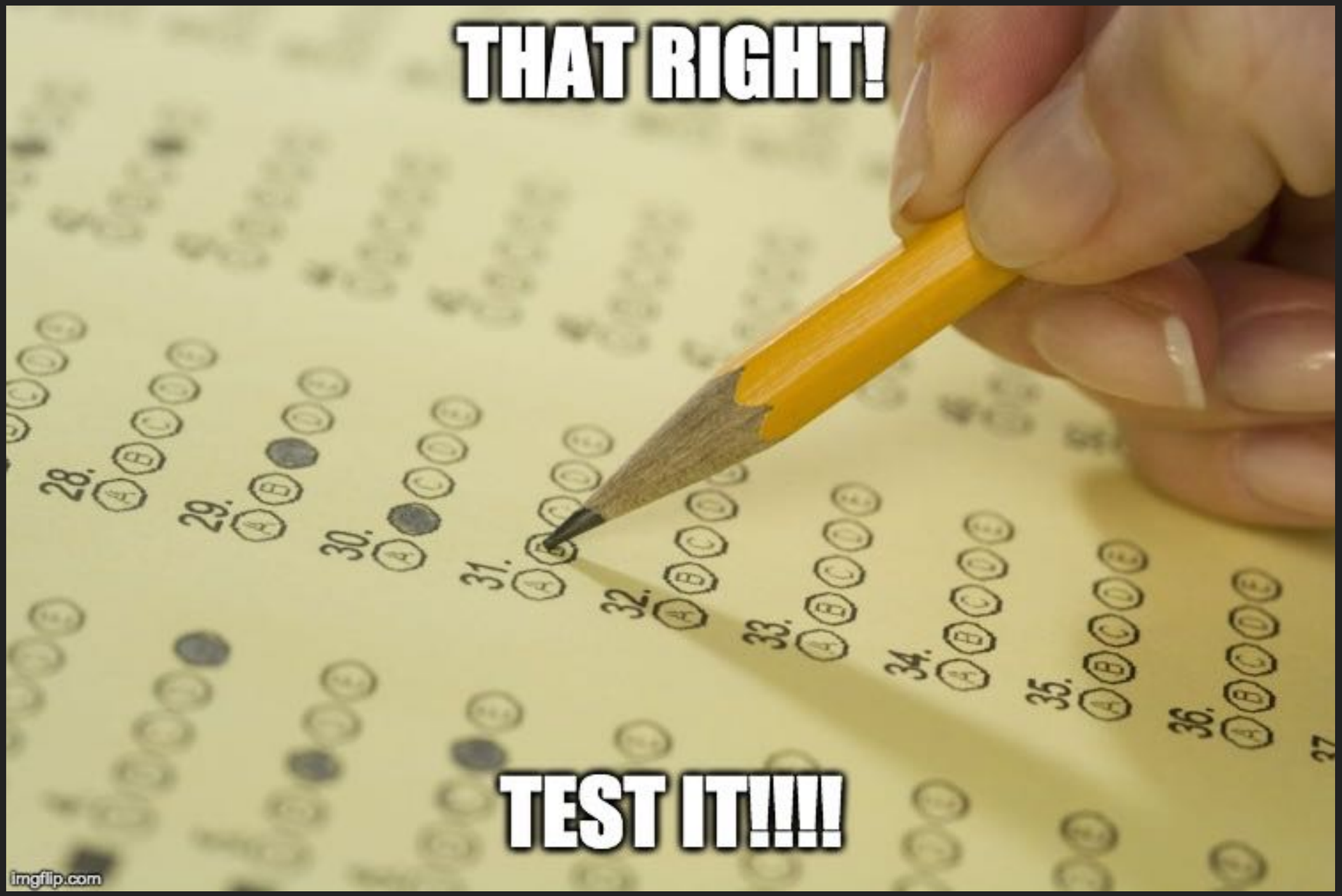
- Test it… but this time you're going to have to use POSTMAN to make this delete request.

REACT EXPRESS & MYSQL

- Okay… we're almost there! Now we have to create a route that will add an employee. As well as update one. These are pretty similar routes. But a lot more is going on. First we need to set a variable that will catch the req.body so we know what data is being added. Then in the query - We define the sql variable that has this query setting all the fields we specified in our schema, then it will call a procedure that we defined in our DB as well. Then - We use that query and fill in all the data that we pull off the "emp" variable… Phew…

```
66  app.post('/employees', (req, res) => {
67      let emp = req.body;
68      let sql ='SET @EmpID = ?;SET @Name = ?;SET @EmpCode = ?;SET @Salary = ?; \
69      CALL EmployeeAddOrEdit(@EmpID,@Name,@EmpCode,@Salary);';
70      mysqlConnection.query(
71          sql,[emp.EmpID, emp.Name, emp.EmpCode, emp.Salary],
72          (err, rows, field) => {
73              if (!err) res.send(rows);
74              else console.log(err);
75          }
76      );
77  });
```

- Guess what?????? When you create a new route… you test it.



- Now for our last one… we need to update a specific employee. This is a lot like the put route that will add an employee… So get to code'n.

```
80  app.put('/employees', (req, res) => {
81      let emp = req.body;
82      let sql ='SET @EmpID = ?;SET @Name = ?;SET @EmpCode = ?;SET @Salary = ?; \
83      CALL EmployeeAddOrEdit(@EmpID,@Name,@EmpCode,@Salary);';
84      mysqlConnection.query(
85          sql,[emp.EmpID, emp.Name, emp.EmpCode, emp.Salary],
86          (err, rows, field) => {
87              if (!err) res.send("Updated Successfully");
88              else console.log(err);
89          }
90      );
91  });
```

REACT EXPRESS & MYSQL

- ....
- ....
- ....
- ....
- ....
- That’s right..... jusssstttt
- test it

- Now that you have everything working correctly. We should celebrate - by doing another workshop that will bring all this data, and CRUD functionality to the front end.