REACT EXPRESS AND MYSQL

BY QCC - TECH WORKS

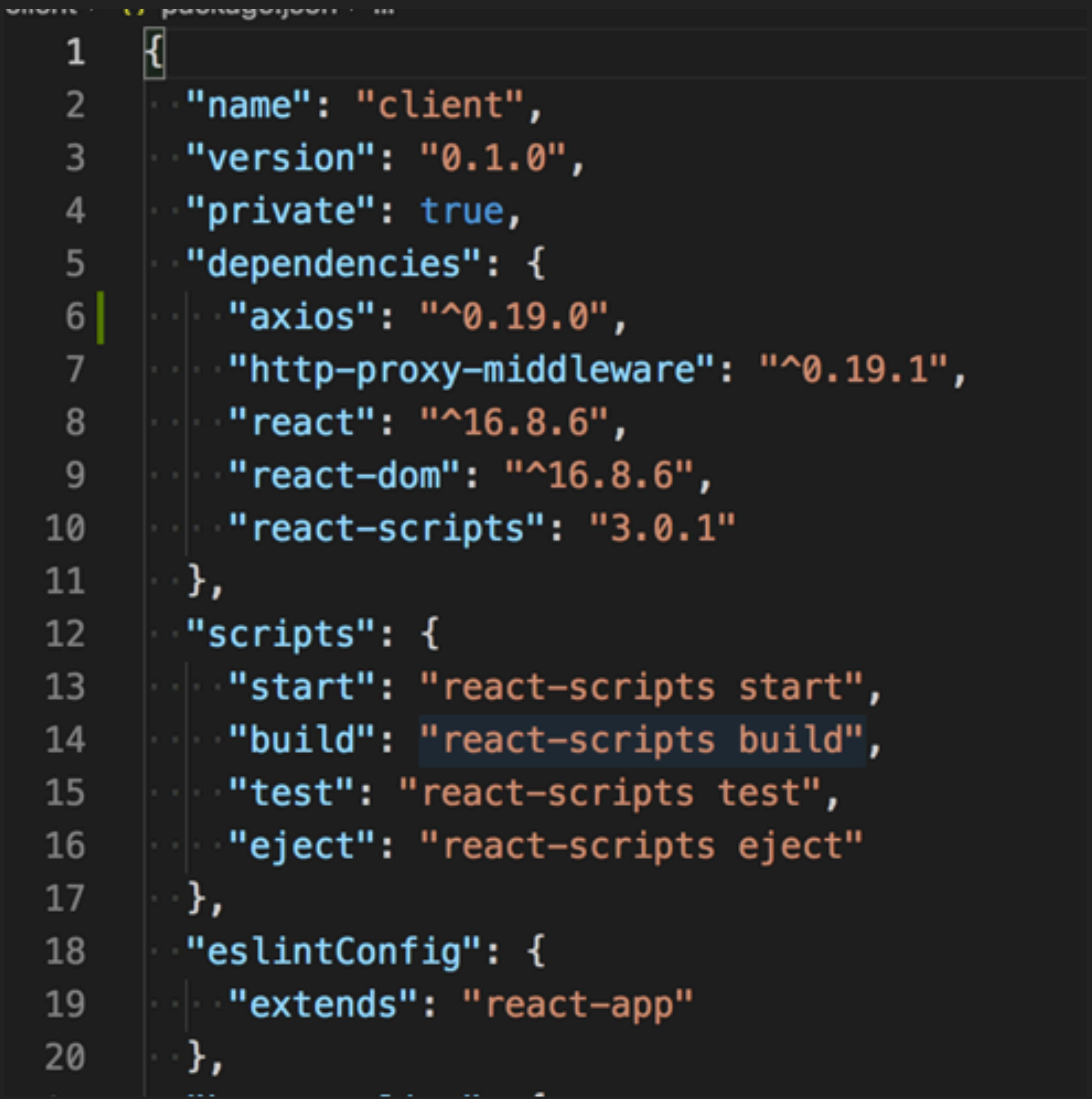
GOAL

- Build on our SQL and Express workshops.
- Connect, and do CRUD operations on a mySQL DB with express.
- Connect React with express and use forms to input date
- Use Axios like a boss.

WHAT IS AXIOS

- Axios is a promise based http request library - which allows us to interface with REST API's.
- Its a lot like fetch… But has some syntactical differences.
- We install this package by opening a command prompt and navigating to our project directory (this will be the main folder that server.js and the client folder are in) and we change directories into our client folder. Then we want to install the specific package by typing "npm install axios"

We should see this dependency listed.

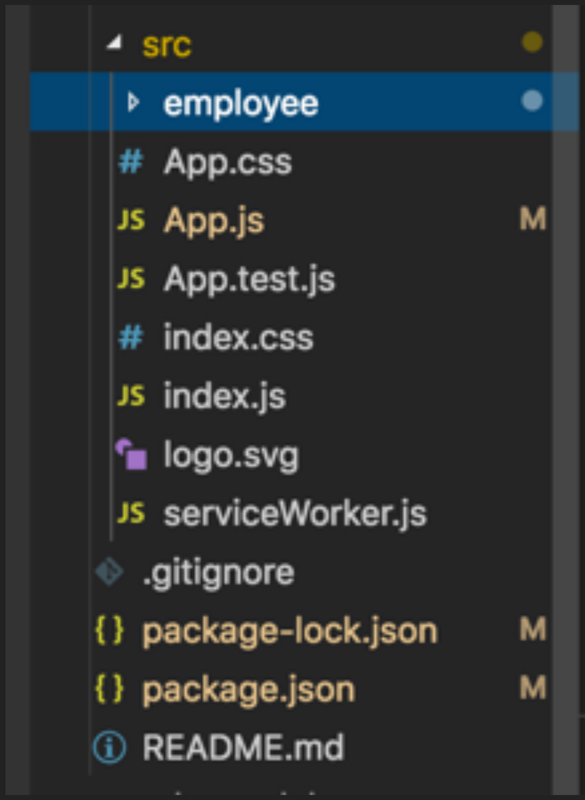


```
{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "axios": "^0.19.0",
    "http-proxy-middleware": "^0.19.1",
    "react": "^16.8.6",
    "react-dom": "^16.8.6",
    "react-scripts": "3.0.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": "react-app"
  },
```

Remember this is in our client directory. We're wanting this package installed on our front end

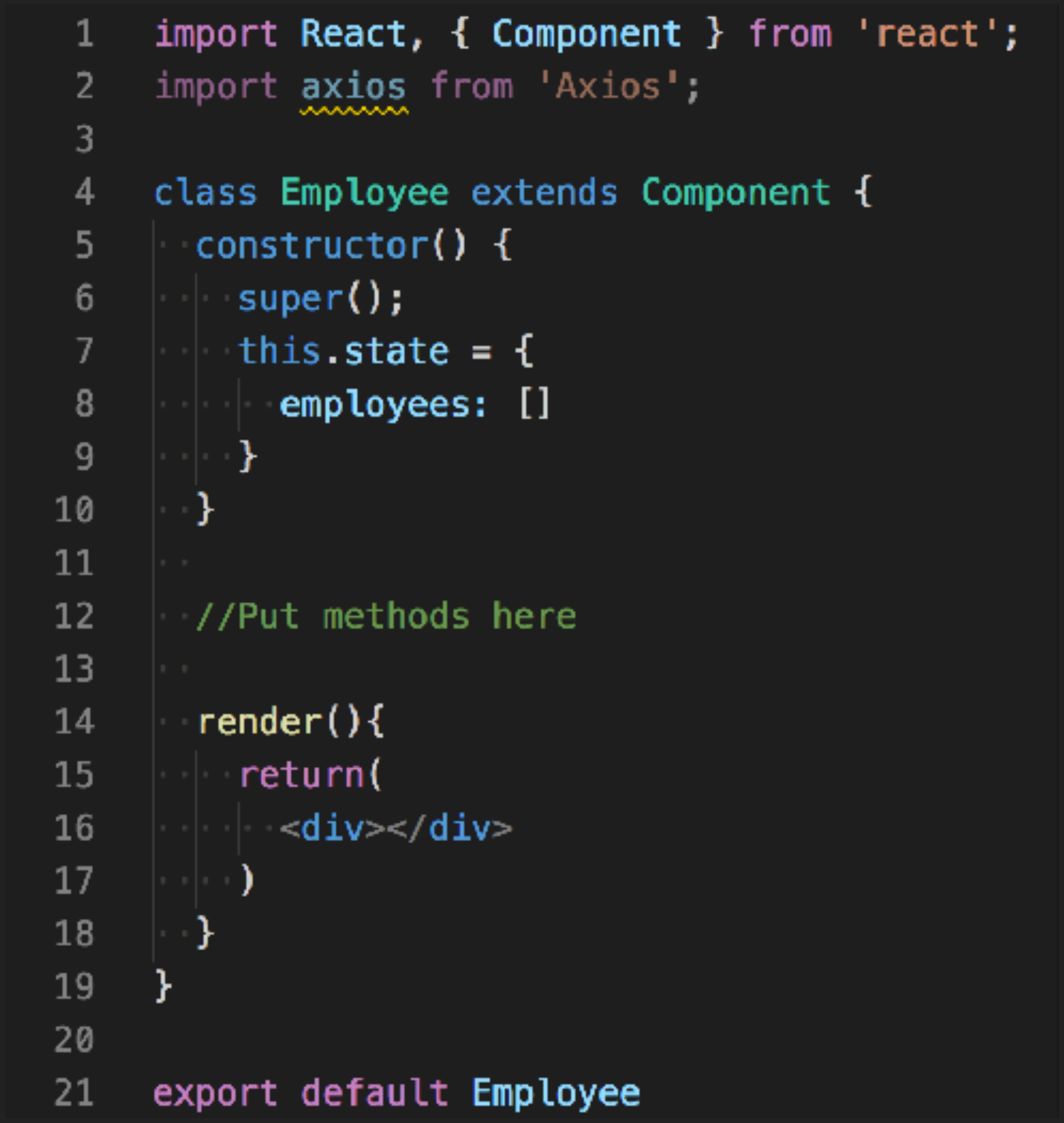
TIME FOR
SOME FRONTEND FUN
imgflip.com

- Lets make it happen. The first thing we want to do is grab all our data and build out a list of that data with a react component.
- Lets first create a new folder in our src directory called employee



- In this folder we also want to create a file called employee.js
- This component will build out our list and also store our methods that we're going to use to do CRUD operations.

- Lets create this component and start coding the base.

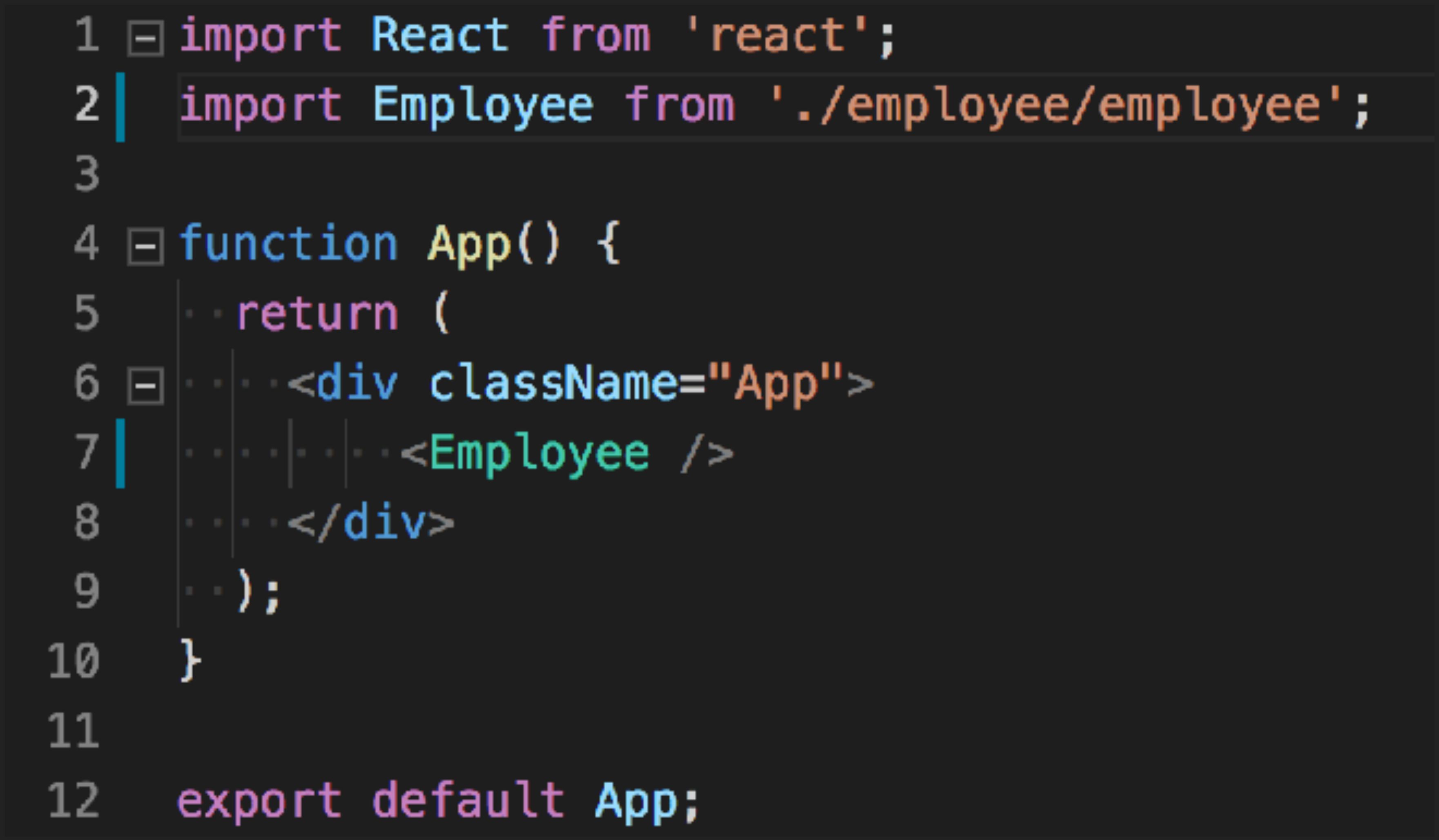


```
import React, { Component } from 'react';
import axios from 'Axios';

class Employee extends Component {
  constructor() {
    super();
    this.state = {
      employees: []
    }
  }
  
  //Put methods here
  
  render(){
    return(
      <div></div>
    )
  }
}

export default Employee
```

- Next, lets bring this component into our main APP.js file. Feel free to make this main page your own and change it how you like. STYLE AWAY!!!!



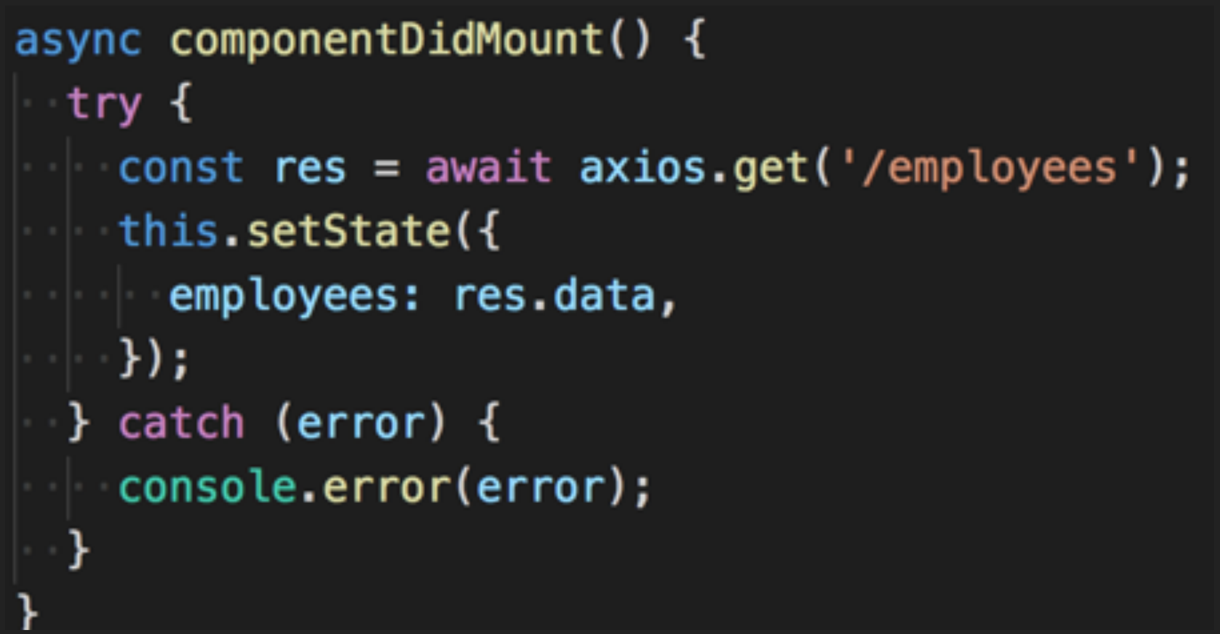
```
import React from 'react';
import Employee from './employee/employee';

function App() {
  return (
    <div className="App">
      <Employee />
    </div>
  );
}

export default App;
```

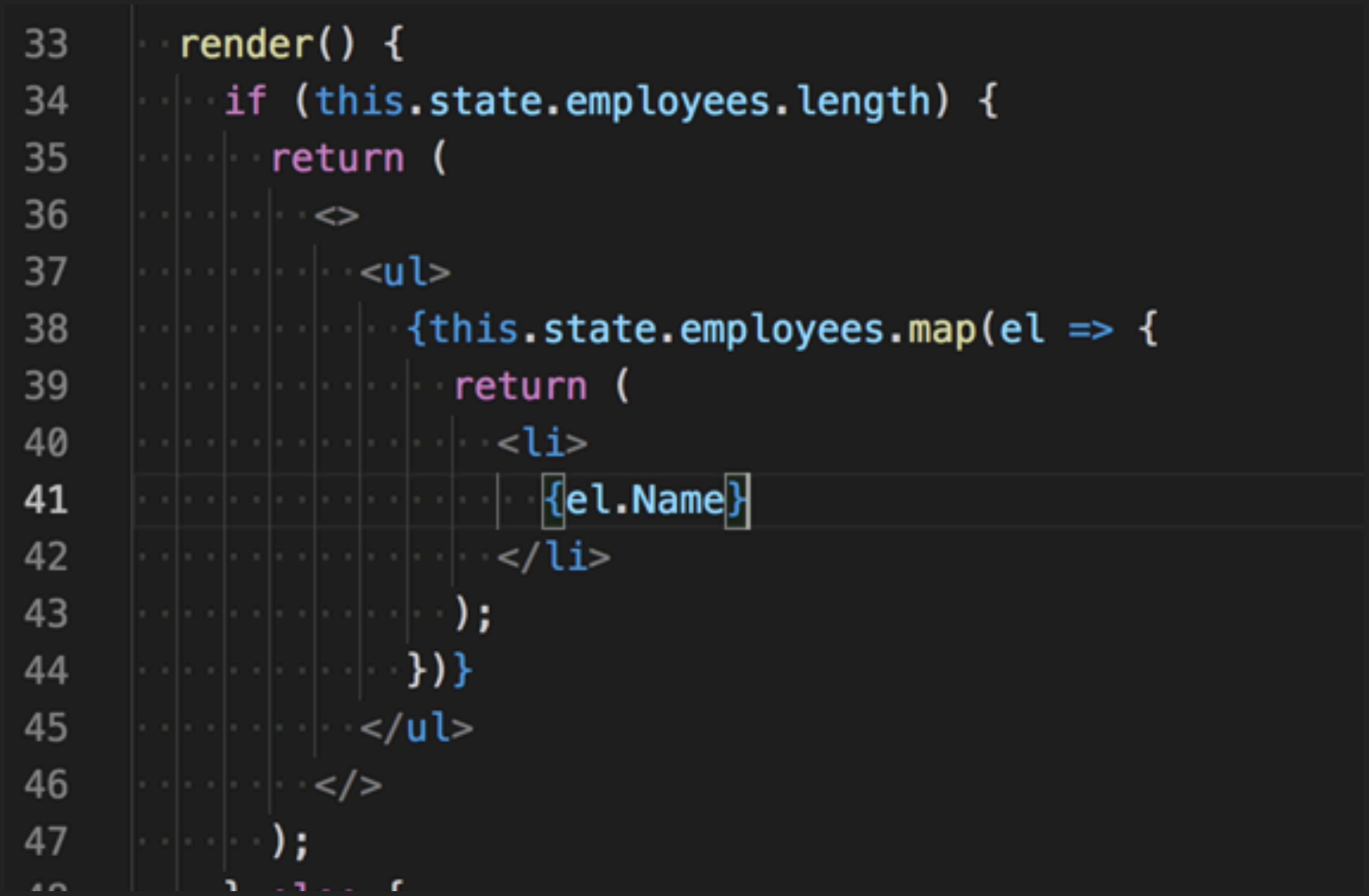
- Okay - Now that our express backend is running like a well-oil'd machine. We can now start building out our front end to interface with our REST-ful API. First thing we want to do is grab all the data - We're going to make a get request that will let us populate a list of the data (the employees in this case).
- Set this up by using a lifecycle method so when the component loads it will use axios to make a request for all our data.

- We're going to use the ES6 async / await syntax to clean these promises up. We're also going to make a get request with axios - grab the response, and then set that data into state.



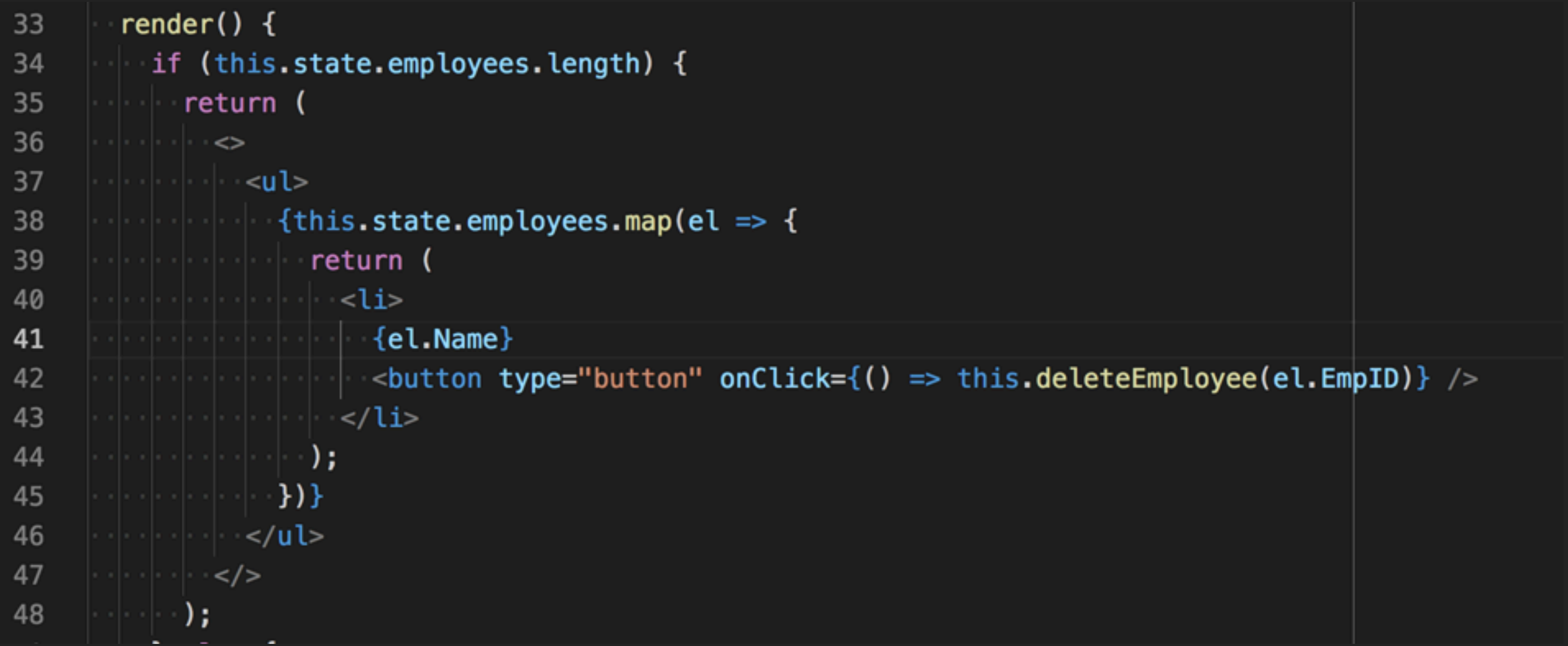
```
async componentDidMount() {
  try {
    const res = await axios.get('/employees');
    this.setState({
      employees: res.data,
    });
  } catch (error) {
    console.error(error);
  }
}
```

- Now if we don't see any errors in our console (remember we're working on the front-end so we can check the browsers console for any errors in our React app page), we know that the request was fulfilled and we can actually map through this state. Don't forget about conditional rendering.



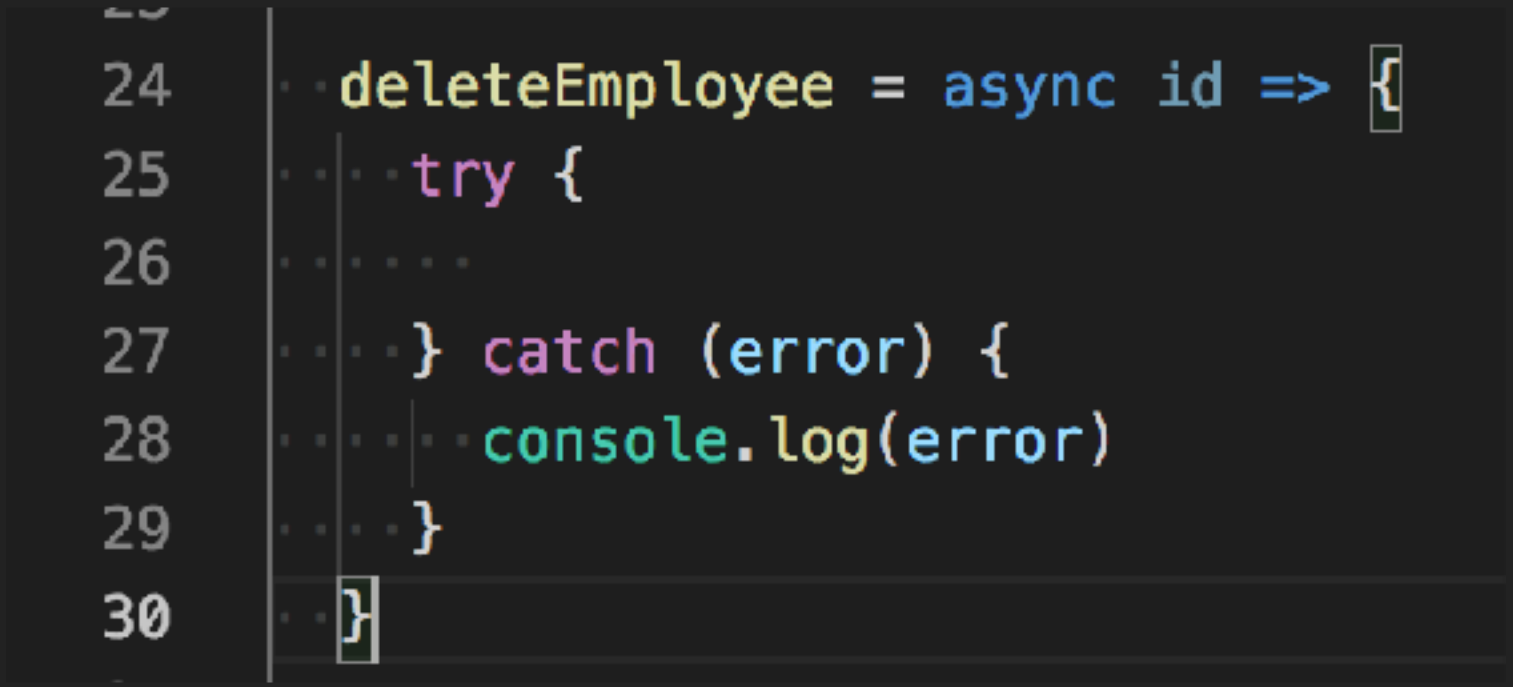
```
  render() {
    if (this.state.employees.length) {
      return (
        <>
          <ul>
            {this.state.employees.map(el => {
              return (
                <li>
                  {el.Name}
                </li>
              );
            })}
          </ul>
        </>
      );
```

- Now we want to start building out all the other functionality that will allow us to CREATE, UPDATE, and DELETE.   First lets tackle the delete functionality. We can simply add a button next to the list item that will give us the functionality to delete that individual item.



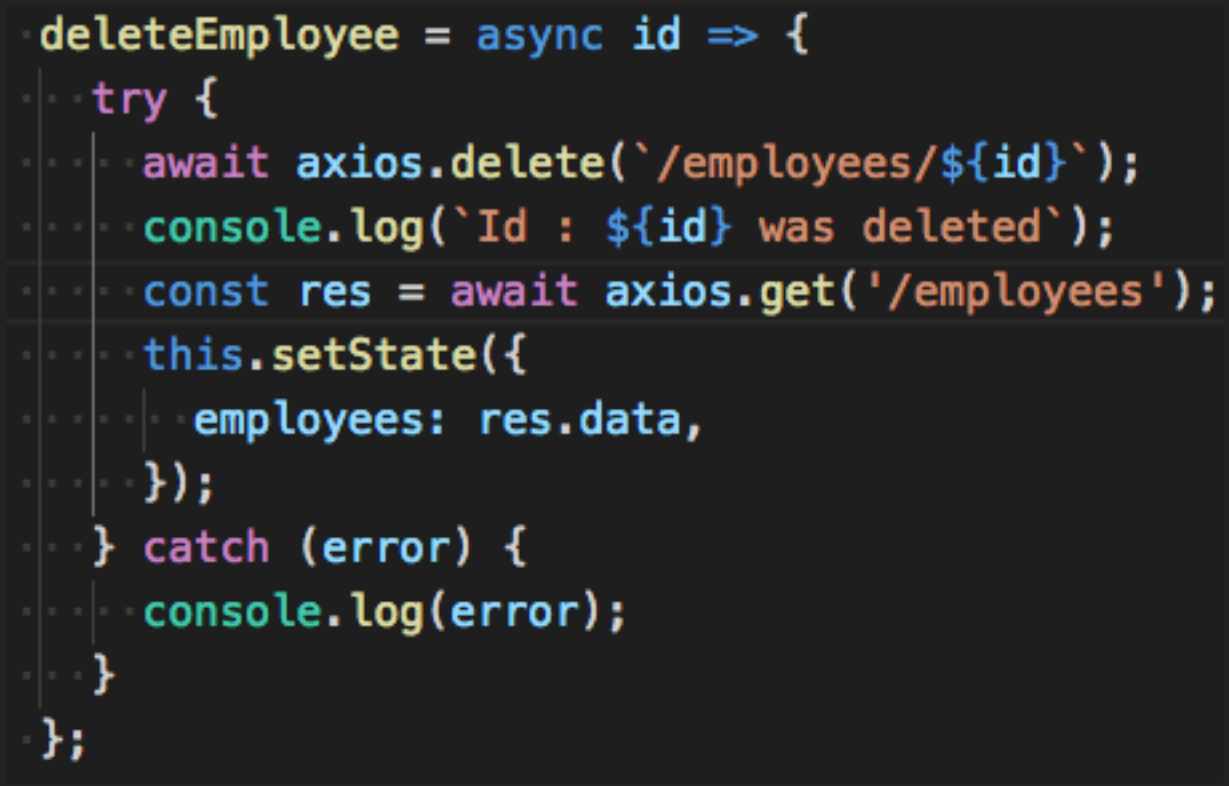
```
  render() {
    if (this.state.employees.length) {
      return (
        <>
          <ul>
            {this.state.employees.map(el => {
              return (
                <li>
                  {el.Name}
                  <button type="button" onClick={() => this.deleteEmployee(el.EmpID)} />
                </li>
              );
            })}
          </ul>
        </>
      );
```

- So the method we attached to the onClick event listener should give you an idea of what to create. I want you to create a method that is asynchronous and will use axios.delete to remove the correct id that corresponds to the list item we click'd the delete button on. Here is a start.



```
  deleteEmployee = async id => {
    try {
      
    } catch (error) {
      console.log(error)
    }
  }
```

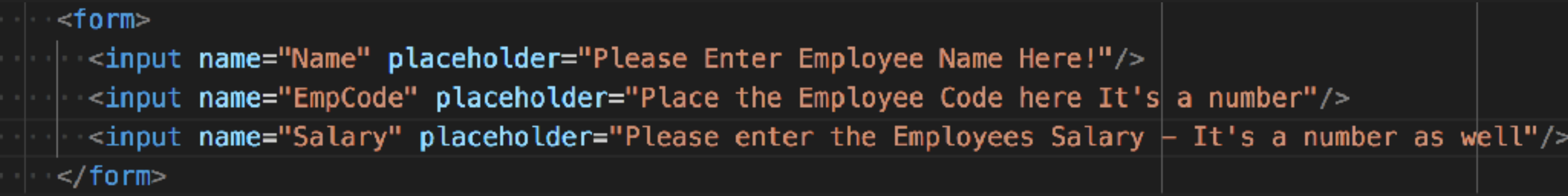
- Here is how I'm accomplishing this task. I'm passing in the ID of the employee we clicked the delete button on, and using that ID to make a axis.delete request using template literals to add in the unique value of the id.



```
deleteEmployee = async id => {
  try {
    await axios.delete(`/employees/${id}`);
    console.log(`Id : ${id} was deleted`);
    const res = await axios.get('/employees');
    this.setState({
      employees: res.data,
    });
  } catch (error) {
    console.log(error);
  }
};
```

- Now lets work on creating a new employee to be added to our DB. First thing is we're going to need some input from the user. The main fields I want from them is the Name - Employee Code & their Salary.
- How are we going to get this from the user???

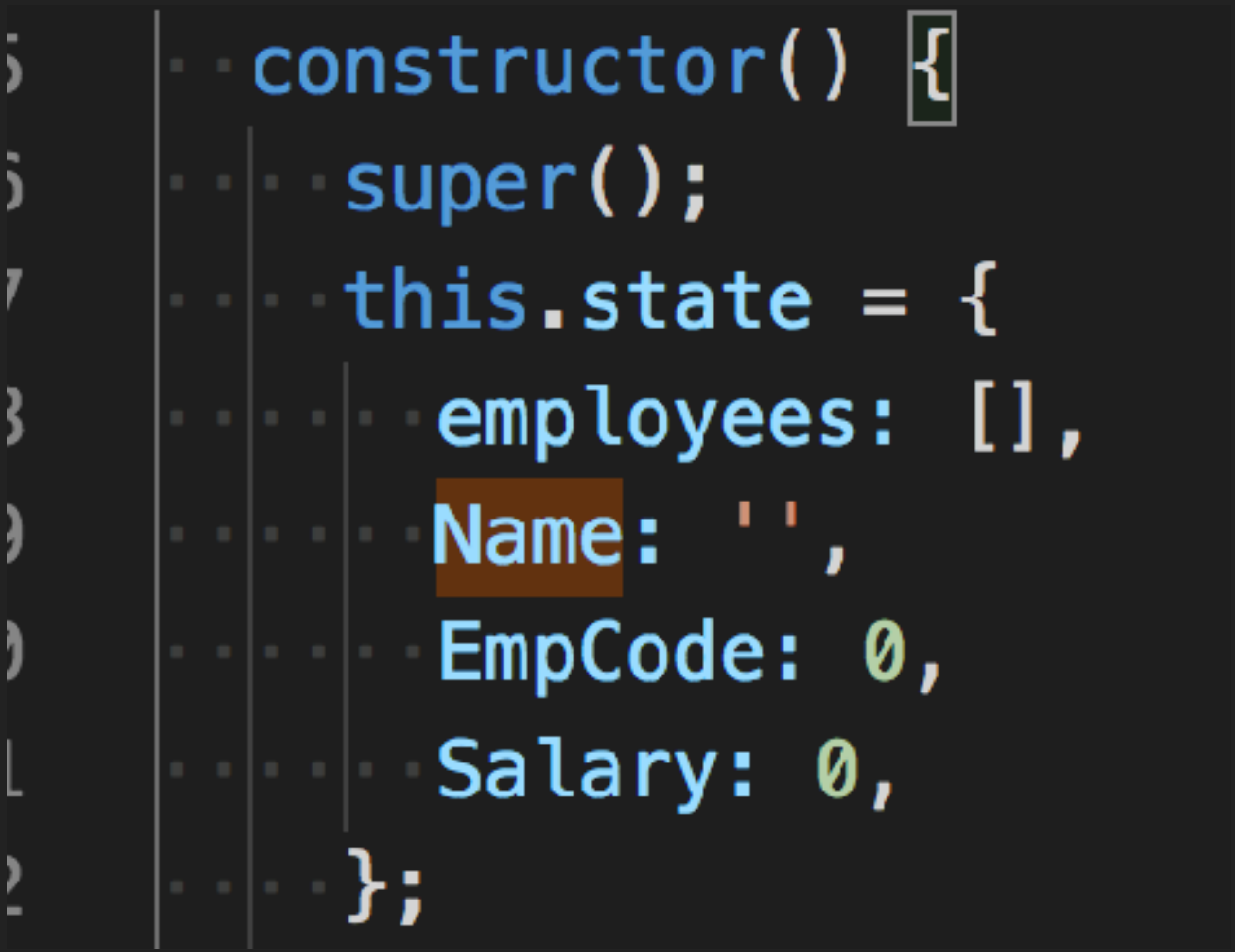
- We're going to build a form with inputs… Lets put this below our unordered list. Lets set it up like this.



```
<form>
  <input name="Name" placeholder="Please Enter Employee Name Here!"/>
  <input name="EmpCode" placeholder="Place the Employee Code here It's a number"/>
  <input name="Salary" placeholder="Please enter the Employees Salary - It's a number as well"/>
</form>
```

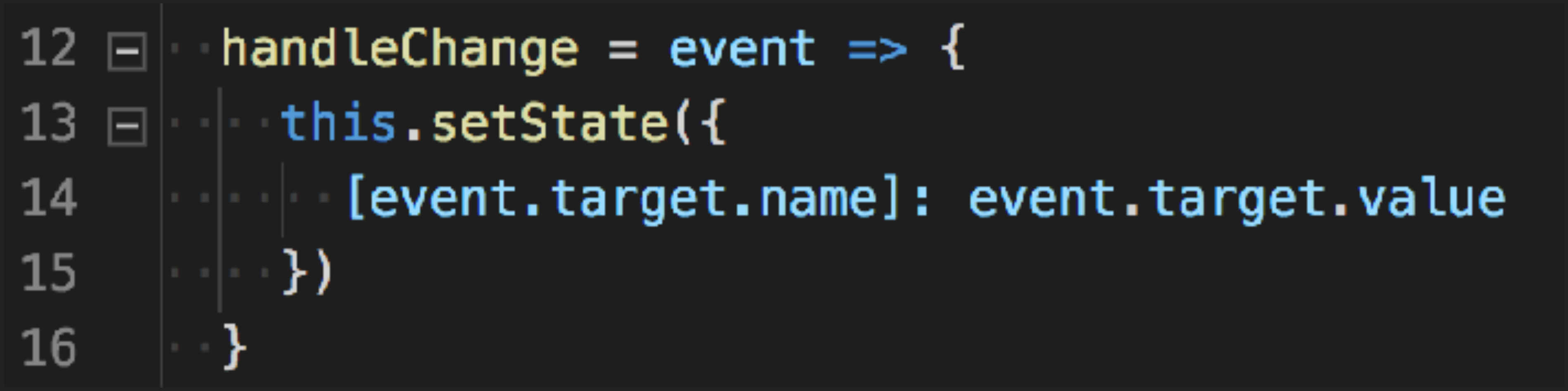
- With forms we need to be able to submit it. So we need to add a onSubmit event handler. The inputs should also be able to handle changes. So if each input is going to handle a change what should we do with state? I want to have values that corresponds to each input.
- Lets test somethings out in repl

- Just like on our todos list we needed to add the values to state that are going to be specific for each input. I'm going to name these exactly the same name as what I assigned the input html name property.



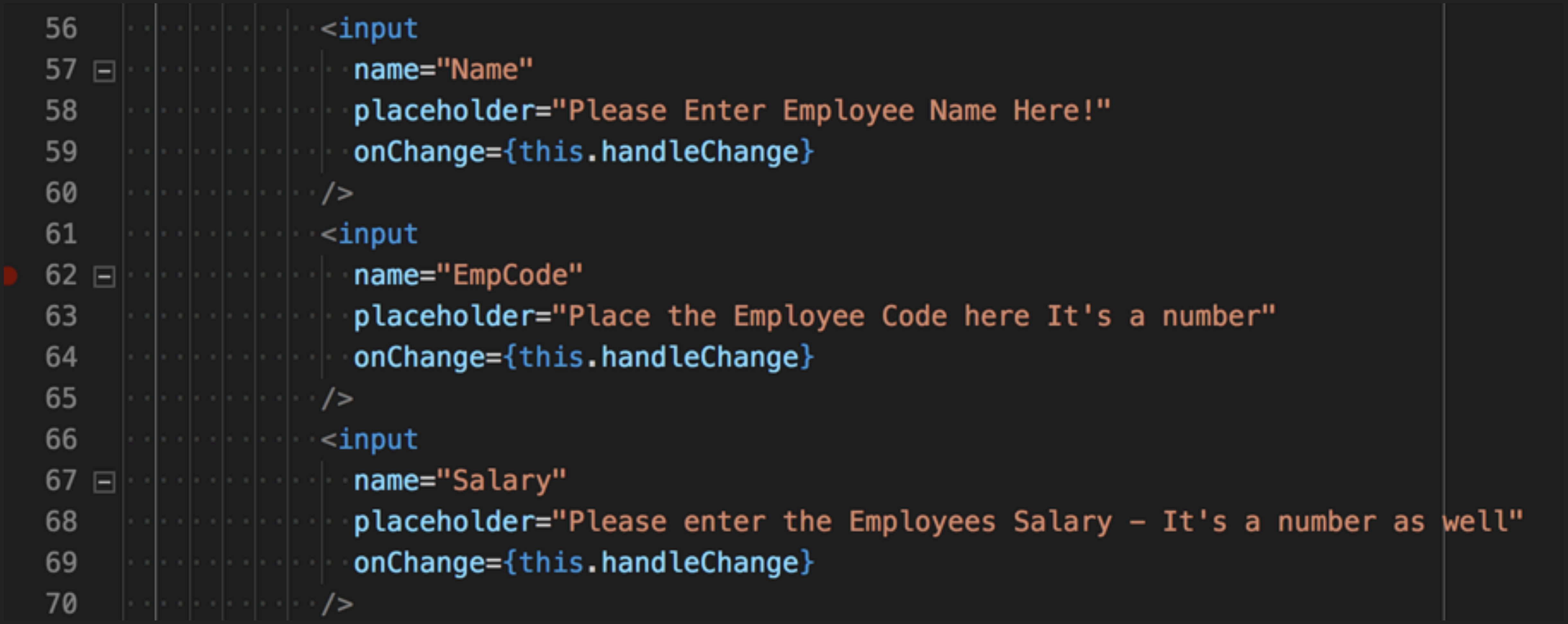
```
  constructor() {
    super();
    this.state = {
      employees: [],
      Name: '',
      EmpCode: 0,
      Salary: 0,
    };
```

- Now with your newfound object knowledge we should be able to bring in a dynamic value for each input. These inputs have an associated label for name that is a string. So what we can do in our handleChange function is bring all these dynamic values and build a single handle change function that can be used by all the inputs.



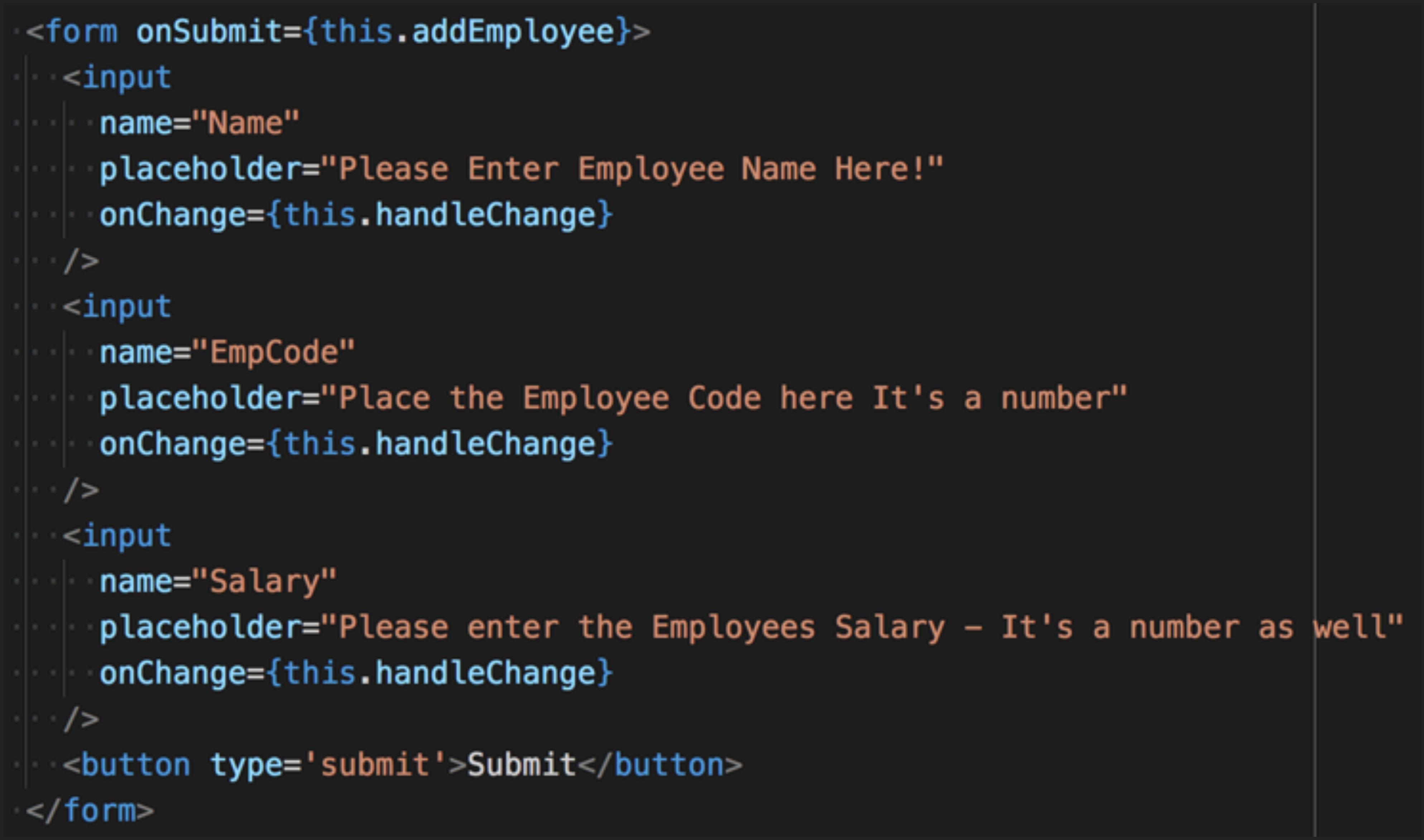
```
  handleChange = event => {
    this.setState({
      [event.target.name]: event.target.value
    })
  }
```

- So we can now make sure we attach our onChange event handler to our inputs



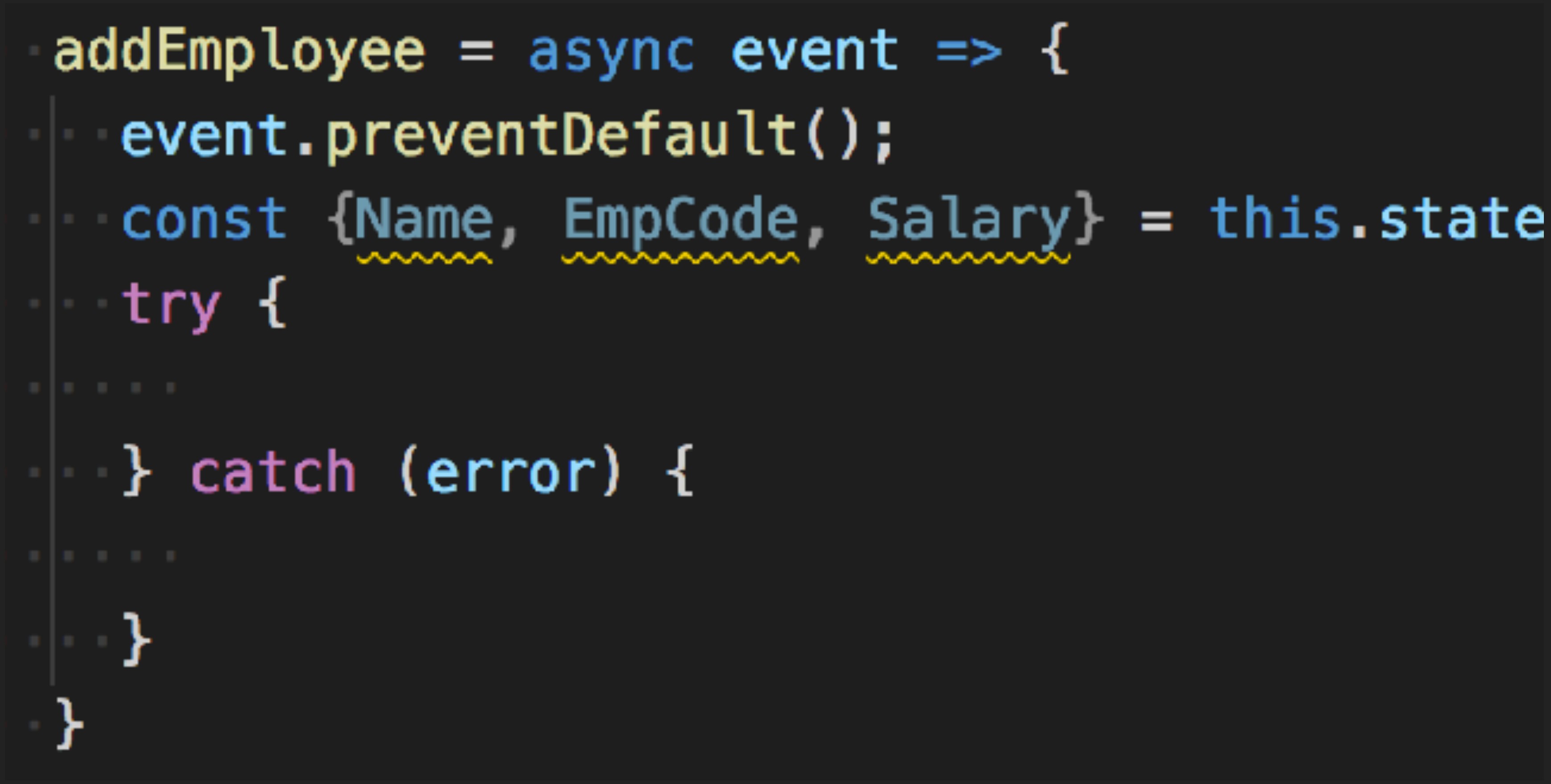
```
<input
  name="Name"
  placeholder="Please Enter Employee Name Here!"
  onChange={this.handleChange}
/>
<input
  name="EmpCode"
  placeholder="Place the Employee Code here It's a number"
  onChange={this.handleChange}
/>
<input
  name="Salary"
  placeholder="Please enter the Employees Salary - It's a number as well"
  onChange={this.handleChange}
/>
```

- Now we want to handle the submit of the form. This is where we're going to call our addEmployee Method - and add a button to submit the form.



```
<form onSubmit={this.addEmployee}>
  <input
    name="Name"
    placeholder="Please Enter Employee Name Here!"
    onChange={this.handleChange}
  />
  <input
    name="EmpCode"
    placeholder="Place the Employee Code here It's a number"
    onChange={this.handleChange}
  />
  <input
    name="Salary"
    placeholder="Please enter the Employees Salary - It's a number as well"
    onChange={this.handleChange}
  />
  <button type='submit'>Submit</button>
</form>
```

- Now lets build out the addEmployee method… try to build it out on your own. I'll give you a head start.

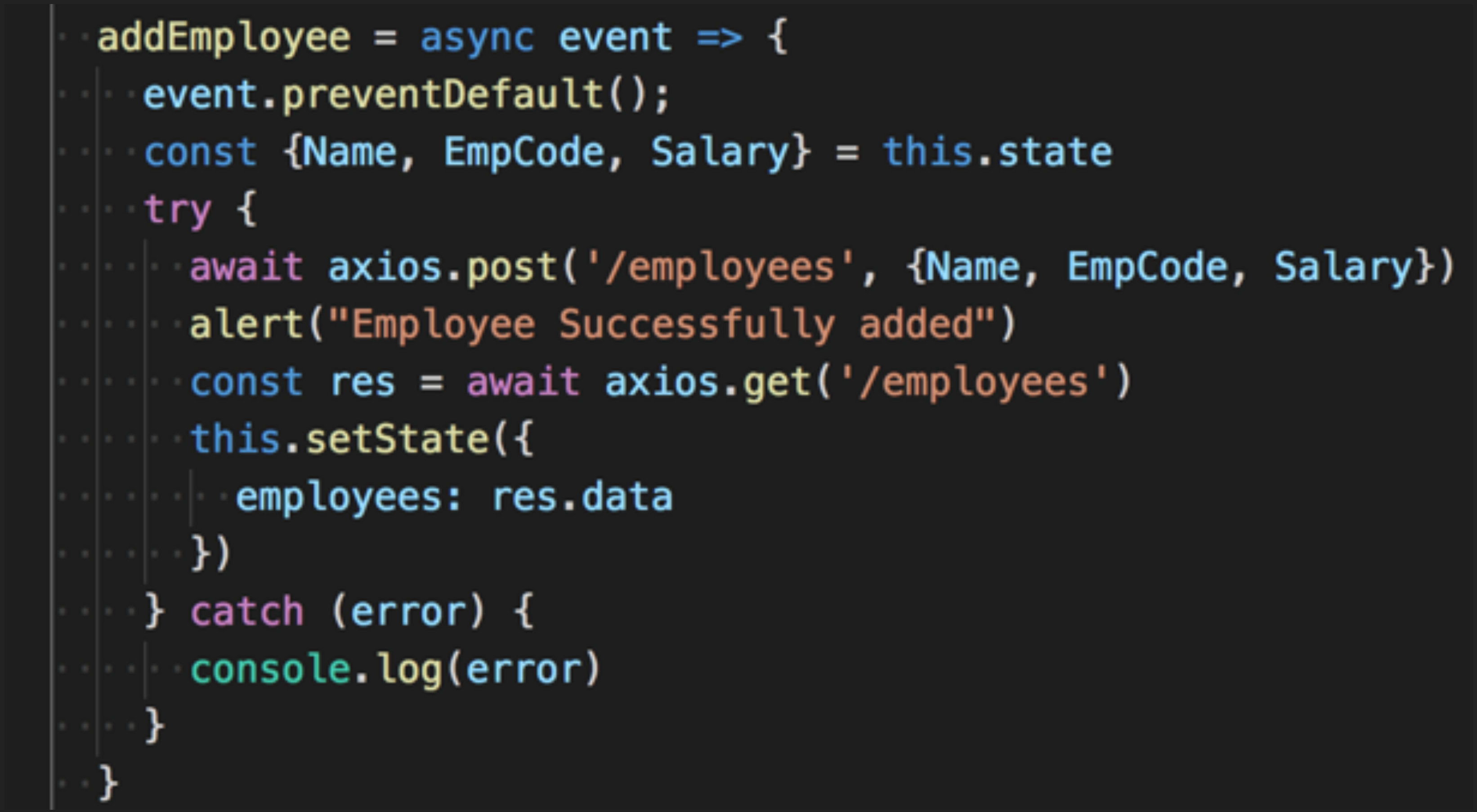


```
addEmployee = async event => {
    event.preventDefault();
    const {Name, EmpCode, Salary} = this.state
    try {

    } catch (error) {

    }
}
```

- I'm doing a post request with some data as the second argument of the post request… then i'm sending an alert to make sure the user knows it updated… then I'm making another git request to refresh our list and set the state to the new list.



```
addEmployee = async event => {
  event.preventDefault();
  const {Name, EmpCode, Salary} = this.state
  try {
    await axios.post('/employees', {Name, EmpCode, Salary})
    alert("Employee Successfully added")
    const res = await axios.get('/employees')
    this.setState({
      employees: res.data
    })
  } catch (error) {
    console.log(error)
  }
}
```

REACT EXPRESS & MYSQL

- Now - If everything is working correctly. Time for you to hack out the update functionality.
- Once you get that finished create a functional component that will handle all the adding and editing of employees.
- Add another functional component that will return all the JSX we need to make the list.
- Then add in the functionality and correct JSX to handle if someone wants to search by ID and will only display that employee.
- Separate our Express routes into another directory into another file
- Then add some styling.