**React Frontend, Node Backend CRUD**

**Code:** [**https://github.com/benjaminrittenhouse/CIS4282-tutorials/tree/main/react-full-crud**](https://github.com/benjaminrittenhouse/CIS4282-tutorials/tree/main/react-full-crud)

1. Now, we will build on top of the code we have that only displays users. This web app will now be able to create (insert), update (edit), delete, and read (display / view). This is also known as **CRUD**.
2. First, let us look at the updated **backend (**server side Node.js)
   1. In the server, you will see a folder **webUserApis**
   2. Within this folder, we divided the APIs into
   3. readAPI
      1. This has calls that allow us to read all the users (which we have seen before, listAllUsers)
      2. And get Role types for a drop down menu we will see in a moment
   4. modifyAPI
      1. This has calls that allow us to create (insertUser) and update (updateUser)
         1. **insertUser**
            1. The insert user function creates empty web user objects that are going to be populated by the webUser information itself, as well as any error messages based on user input or formatting errors
            2. It is initially populated with the object passed from the front end, using **req.query.** Req.query can take an object through the api URL and take out variable values, such as:

[**http://localhost:5000/api/insertUser?webUserId=&userEmail=asdf&userPassword=asdf&userPassword2=asdf&image=asdf&birthday=2020-05-05&membershipFee=100&userRoleId=asdf&errorMsg=**](http://localhost:5000/api/insertUser?webUserId=&userEmail=asdf&userPassword=asdf&userPassword2=asdf&image=asdf&birthday=2020-05-05&membershipFee=100&userRoleId=asdf&errorMsg=)

* + - * 1. For example, req.query.userEmail would give us **asdf**
        2. We then check for errors using our **DbUtils** created in a previous tutorial. If all is good, we do some error checking and decide whether to send back error messages or **a success message** indicating that the web user was inserted.
      1. **updateUser**
         1. Update user follows many of the same principals of insertUser, however the SQL statement changes to update vs creating a new record. All error checking is still done, and we are still using req.query.

1. Now that we saw the server side, let us check out where this code is being called on the **frontend.** It is going to be more of our **asynchronous methods** that we saw in listAllUsers before.
   1. Navigate to the client side folder.
   2. Under **src/components/webUser,** notice the following files:
      1. **Display,** which we saw before that lets us listAllUsers
      2. **Insert.js**
         1. Insert allows us to create a new record
         2. To start, we created state variables to again keep track of user objects how we similarity did on the backend (we are passing objects between the two to keep track of the data)
         3. Notice the async function that is making a call back to the API, but this time it is attaching an object to the end of the URL:

const str = `${process.env.REACT\_APP\_API\_URL}/api/insertUser?${objToStr}`;

* + - 1. This is what we mentioned earlier, where req.query grabs the object fields out of the URL itself.
      2. The div being returned is simply a list of input fields that utilize state and our function to create a user and send it to the backend
    1. **UpdateModal.js**
       1. UpdateModal is very similar to insert with the state variables, the asynchronous function, but this time we display it in a modal on the frontend vs an entirely separate page. That way we aren’t jumping between pages so often.
    2. **DeleteModal.js**
       1. DeleteModal is a component that displays on the page upon clicking the **delete** button next to each respective user. To ensure that you meant to delete the user, there is an **OK** and a **Cancel** button that will appear on the modal window
       2. When OK is clicked, the frontend will call the deleteUser API with the selected user’s webUserID.
       3. If Cancel is clicked, the modal window is hidden and the API call is not invoked.

1. **URL Tampering**
   1. There are React front end pages to do all of the CRUD listed above. Also notice the links on the navbar that are underlined, allowing you to do API tampering without the frontend.
      1. “ListAllUsers” simply lists the JSON of users when the list all API is called.
      2. “InsertUser”
         1. Move to App.js in your client side folder.
         2. Note the userData state variable created. Alter this how you please, and then use the “insertUser” link to call an insert with this object. Tamper with it how you would like to create errors you may imagine with “wrong” objects.
      3. “deleteUser”
         1. deleteUser will attempt to delete whichever user has the web\_user\_id at the end of the URL. You can test different ID’s simply by changing this number at the end.
         2. If the record is successfully deleted, the JSON object will contain the message saying that the user was deleted. If the record was not deleted, the JSON object will have an error message.
      4. “updateUser”
         1. Update remains similar to insert as we have seen. However, to tamper with this URL, please see the updateUserObject that has fields already created. Be sure you make the webUserId match a record that is already in your database, and then change something that you would like to update.
         2. Call listAllUsers and you should see the change reflected, based on the JSON response you got (success vs. error)