**Node.js ListAllUsers WebAPI Tutorial w/ React**

1. Now, we would like to take our single page application and add some links that reference a **web API** that fetches some user data for us. For the time being, let us have these links simply call the API and return the JSON.

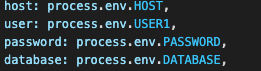
**Code:** <https://github.com/benjaminrittenhouse/CIS4282-tutorials/tree/main/listAllUsers-project>

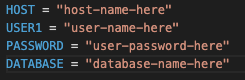
**Before starting,** another note on environment variables.

* Create another **.env** file within the **client** side of the project.
* #.env file
* REACT\_APP\_API\_URL = http://localhost:5000
* Place this code within the .env file. This is so we can clarify the link we are referencing and do not have to write it every time in all places that it changed.
* Then, go to App.js within the client side and note the link on line 24 where we use the environment variable:

const api = process.env.REACT\_APP\_API\_URL + "/api/listAllUsers";

1. We should be familiar with the client side / frontend by now. What is new in this tutorial is using the **Node** backend to fetch some database information.
   1. Within the code above, navigate to the **server** folder, then to **dbUtils/DbConn.js**
      1. Notice that there are references to environment variables within the file. These help in hiding sensitive information when using version control such as GitHub. The references look like this:



* + 1. To create your own environment variables, within the **server** folder, create a file that does not have a name, and only says “**.env”.** To be clear, that is a file that only has the .env extension, no characters before it.
    2. Fill in the .env file as such, with your Heroku DB information that you should have retrieved when setting up Heroku in a previous tutorial:
       1. 
       2. You can call these variables whatever you would like. I believe USER is a keyword sensitive variable name on MacOS, so I simply added a 1 to the end, but this is up to preference.
  1. You should be connected to your database now. Let us move on to running locally.

1. Because we now have a **frontend (React)** and a **backend (Node.js),** we must have both of them running at the same time. First, within your terminal or command line, **run the react app** how we have before (npm start).
   1. **Next,** to run the back end, **open another terminal / command line window.** On Mac you can do this by CMD+T to open a new one.
   2. Change directories out of the front end folder, by doing **cd ..** until you are back to the parent directory. Then, cd into **server.** From here, we will use the following command to run the **backend:**

**nodemon server.js**

* 1. Nodemon is a package that allows us to run the backend without having to re-run it when we make changes!

1. **With both your frontend and backend running,** navigate to your browser and ensure the React app is visible. Click the link in the **Navbar** that references the webApi to see the JSON returned from your database!
   1. Note two links: listAllUsers, and getById. These are two URLs referencing our server that will show us JSON. If the getById is giving an error or not returning any JSON, reference the instructions below for more details (Understanding the Code)
2. **Understanding the code**
   1. Let us make sense of the web API. We created our backend folder, and in that folder we connected to our database.
   2. The server itself is running on **server.js.** Open that file.
      1. You will notice a bunch of **require** statements, essential to getting Node.js to run. More importantly, note the **const db = require()** statement, where we import our database!
   3. We create our express Node app, and now we can create routes to our API. We created a **“/” route** that does not do anything but console log.
   4. **app.get(“/api/listAllUsers”)** is what we are looking for. In this get statement…
      1. We create a variable that stores the SQL statement to fetch our users, joined with the roles table, and sorted accordingly.
      2. Then, using our **db** variable that references our database from earlier, we use the **.query()** method to use our sql statement and get the results as JSON from the database.
      3. Finally, to send it to the frontend or back to the user, we use **res.send(),** referencing res from the beginning of our function parameters.
         1. Note part 6 below about the other parts of the APi call, such as formatting of webUser
      4. This allows us to view the JSON on the frontend via the localhost URL.
   5. **app.get('/api/getUser/:id”).** In this get statement…
      1. We are using the URL to fetch a user by an ID. For example, if we navigate to /api/getUser/504, it would get us (if it exists) a record with the web\_user\_id of 504. This is a simple change in the SQL statement within this .get()
      2. To tamper with this url, either edit it in your browser to match an existing web\_user\_id that is in your database, or go to App.js on the client side, and you will see on line 11 that we are creating the URL to tamper with and inserting an ID.
   6. **NOTE:** The localhost of the frontend vs the backend is and should be different. For example, in our case, at the end of our **server.js** on the server side, we **listen** on port 5000. This is just telling the backend where we can locally run things.
      1. Vs. the frontend where we are listening on port 3000
3. **Formatting - WebUser**
   1. While opening the folder **dbUtils,** you may have noticed the formatting and validation utils files. These files are simply used to validate data that is being sent to the database (for when we do insert in the future) and to format data that is being read from the database (for readability and consistency).
   2. We also have a path, **models/webUser/DbMods.js** which utilizes the files mentioned above (validate and format) to our need for a web user. This promotes code reusability and allows us to format and validate any type of data we would like.
      1. For example, you may notice that the membership fee has a dollar sign before the dollar amount when viewing the JSON.
   3. Now, going back to **server.js,** you notice that we are iterating through the JSON that we got from the API call, and formatting that using this DbMods file. Finally, we send the formatted users to the frontend.

**Now,** we can publish via a Git Push to your repository that is hosted on Heroku. If you still need to do this, access the instructions here: <https://devcenter.heroku.com/articles/git#:~:text=To%20deploy%20your%20app%20to,heroku%20main%20Initializing%20repository%2C%20done>