

# SUBMISSION FOR B4

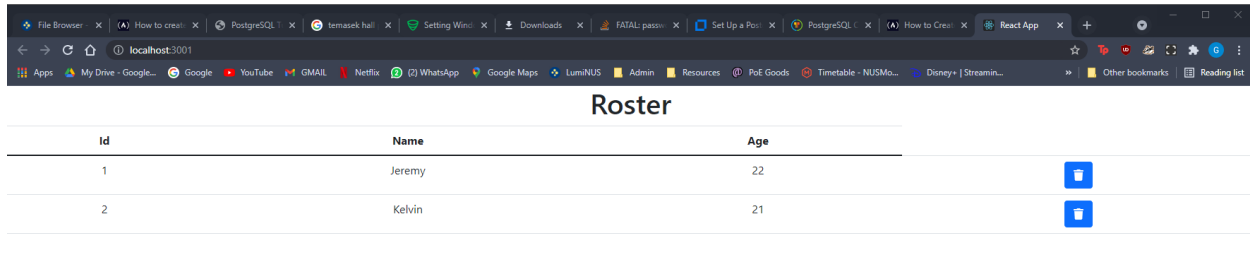
Name: Gordon Foo

Matriculation Number: A0199554L

Github Link: [https://github.com/gordonfgz/CS3219\\_OTOT\\_TASK\\_B.git](https://github.com/gordonfgz/CS3219_OTOT_TASK_B.git)

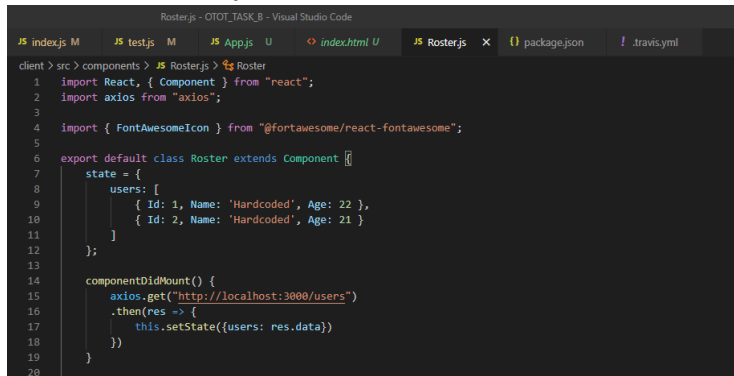
## 1. Using React frontend to interact with backend API

Created a Roster class that queries from a backend server and produces the data on a webpage:



Id	Name	Age
1	Jeremy	22
2	Kelvin	21

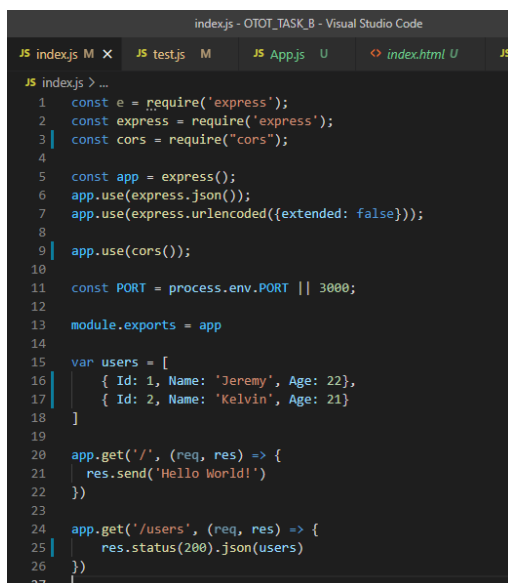
Code lies in Roster.js



```
1 import React, { Component } from "react";
2 import axios from "axios";
3
4 import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
5
6 export default class Roster extends Component {
7   state = {
8     users: [
9       { Id: 1, Name: 'Hardcoded', Age: 22 },
10      { Id: 2, Name: 'Hardcoded', Age: 21 }
11    ]
12  };
13
14  componentDidMount() {
15    axios.get("http://localhost:3000/users")
16      .then(res => {
17        this.setState({users: res.data})
18      })
19  }
20 }
```

If results were not queried properly, then the webpage would display the name as “Hardcoded”.

Evidently, it isn’t hardcoded because the users array in Roster.js is updated by accessing the get api in index.js in the backend by using axios.get()



```
1 const e = require('express');
2 const express = require('express');
3 const cors = require("cors");
4
5 const app = express();
6 app.use(express.json());
7 app.use(express.urlencoded({extended: false}));
8
9 app.use(cors());
10
11 const PORT = process.env.PORT || 3000;
12
13 module.exports = app
14
15 var users = [
16   { Id: 1, Name: 'Jeremy', Age: 22},
17   { Id: 2, Name: 'Kelvin', Age: 21}
18 ]
19
20 app.get('/', (req, res) => {
21   res.send('Hello World!')
22 })
23
24 app.get('/users', (req, res) => {
25   res.status(200).json(users)
26 })
27
```

Here is the index.js that has all the backend apis.

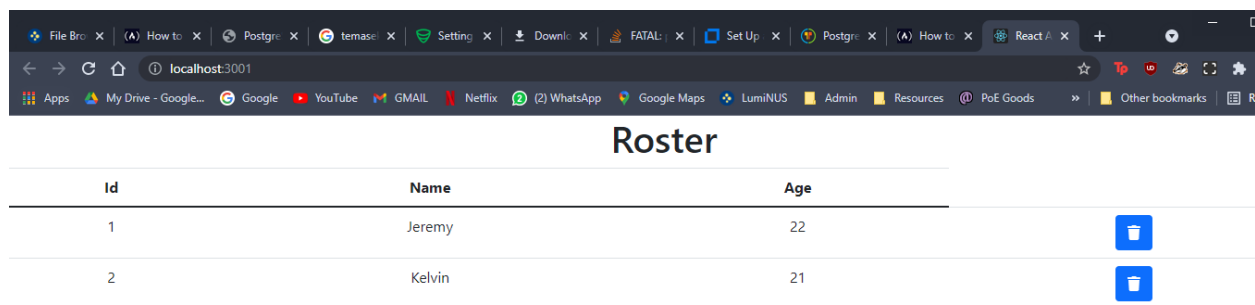
The get request of /users is called in Roster.js and that is how the webpage is able to produce data from the backend.



Roster class is imported into React App:

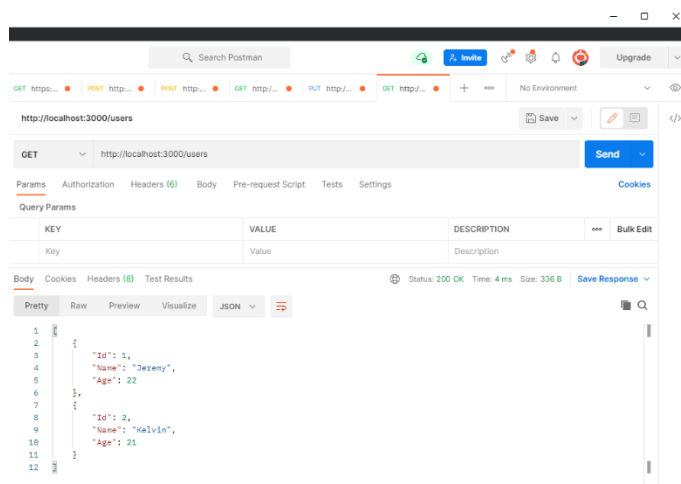
```
client > src > JS App.js > ...
4 import 'bootstrap/dist/css/bootstrap.min.css';
5 import Roster from './components/Roster';
6
7 import { library } from '@fortawesome/fontawesome-svg-core';
8 import { faTrash, faPlus, faEdit } from '@fortawesome/free-solid-svg-icons';
9 library.add(faTrash, faEdit, faPlus);
10
11
12 function App() {
13
14   return (
15     <div className="App">
16       <Roster></Roster>
17     </div>
18   );
19 }
20
21 export default App;
22
```

Delete button in webpage works with the backend:

BEFORE PRESSING DELETE:



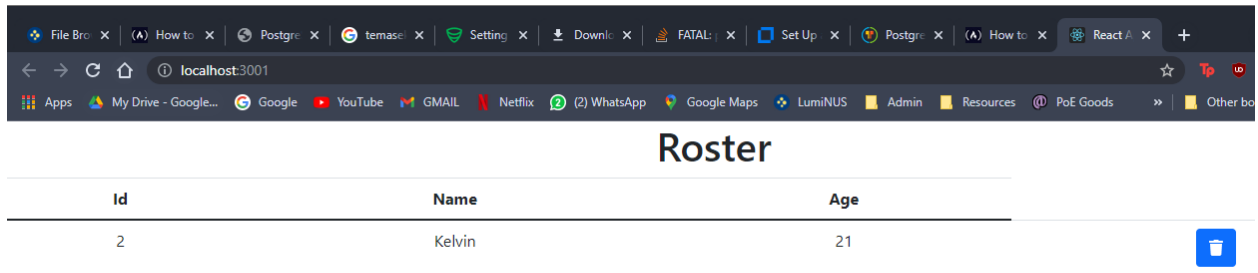
Id	Name	Age	
1	Jeremy	22	
2	Kelvin	21	



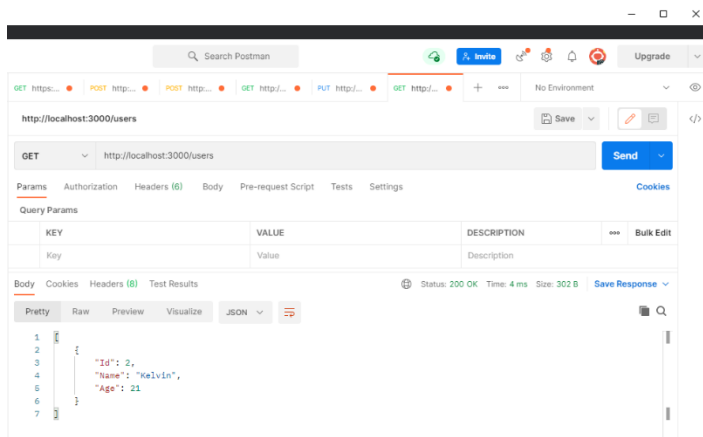
```
1 {
2   "Id": 1,
3   "Name": "Jeremy",
4   "Age": 22
5 },
6 {
7   "Id": 2,
8   "Name": "Kelvin",
9   "Age": 21
10 }
11
12
```

GET request from postman shows that there are 2 entities, inline with what we see in the webpage. Pressing delete on the webpage should result in the next GET request in postman showing only the name that is not deleted. (To prove that deletion happens in the backend and our front end managed to call an API that deals with the data)

AFTER PRESSING DELETE UNDER JEREMY'S NAME:



Id	Name	Age
2	Kelvin	21



As you can see, after pressing the delete button, GET request only returns 1 entry. This means that the delete button on the webpage successfully called an API to update the “database” that an entry is deleted.

## 2.Used bootstrap

```
App.js - OTOT_TASK_B - Visual Studio Code
index.js M test.js M App.js U X Roster.js () package.json ! .travis.yml
client > src > JS App.js > ...
1 import React from "react";
2 import logo from './logo.svg';
3 import './App.css';
4 import 'bootstrap/dist/css/bootstrap.min.css';
5 import Roster from './components/Roster';
6
7 import { library } from '@fortawesome/fontawesome-svg-core';
8 import { faTrash, faPlus, faEdit } from '@fortawesome/free-solid-svg-icons';
9 library.add(faTrash, faEdit, faPlus);
10
11
12 function App() {
13
14   return (
15     <div className="App">
16       <Roster/>
17     </div>
18   );
19 }
20
21 export default App;
22
```

imported bootstrap in App.js

```
index.js M test.js M App.js U index.html U X Roster.js () package.json ! .travis.yml
client > public > index.html > HTML > body
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8" />
5   <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
6   <meta name="viewport" content="width=device-width, initial-scale=1" />
7   <meta name="theme-color" content="#000000" />
8   <meta
9     name="description"
10     content="Web site created using create-react-app"
11   />
12   <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
13   <!--
14     manifest.json provides metadata used when your web app is installed on a
15     user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
16   -->
17   <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
18   <!--
19     Notice the use of %PUBLIC_URL% in the tags above.
20     It will be replaced with the URL of the 'public' folder during the build.
21     Only files inside the 'public' folder can be referenced from the HTML.
22
23     Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
24     work correctly both with client-side routing and a non-root public URL.
25     Learn how to configure a non-root public URL by running 'npm run build'.
26   -->
27   <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-3a90eayHORNB8PW" crossorigin="anonymous">
28   <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-2d8ygl4762+g3779s30d5Y7a9pl0T3mu8Na7C8d7ml3dtufat8p8480a539230">
```

added bootstrap into index

```
Roster.js - OTOT_TASK_B - Visual Studio Code
index.js M X test.js M App.js U index.html U Roster.js X () package.json ! .travis.yml
client > src > components > JS Roster.js > Roster > render > state.users.map() callback
19
20
21
22 deleteUser = (user) => {
23   const filteredItems = this.state.users.filter(x => x.Id !== user.Id);
24   axios.put("http://localhost:3000/users", filteredItems).then(res => {
25     this.setState({users: res.data})
26   });
27 };
28
29
30
31
32 render() {
33   return (
34     <div>
35       <h1>Roster</h1>
36       <table className="table">
37         <thead>
38           <tr>
39             <th>Id</th>
```

used bootstrap styling in Roster.js

Special thanks to:

<https://www.freecodecamp.org/news/how-to-create-a-react-app-with-a-node-backend-the-complete-guide/>

<https://jinalshah999.medium.com/reactjs-step-by-step-tutorial-series-part-4-build-todo-application-using-reactjs-ab219c9b3608>