Lab 05-03 Great Job Lou! 250/250

Bugs Screen: Many of the sortBy options are not returning the correct results.

Suggestion: Probably best to have a label above each of the Sort fields. For example Max age and Min age - Once you start changing them you have to figure out which one is which one. Also the min/max fields allow negative numbers.

horizontal line

# Overview: Issue Tracker

In this assignment we will continue to develop the front-end for our issue tracker with React.

In this assignment we will connect our front-end to our backend and bring some important libraries.

*This assignment is worth 250 points total (or 2.5 lab grades.)*

# Step 1. Use your existing GitHub Repositories

Use your **issue-tracker-react** repository for the front-end code.

And use your **awd1111-issue-tracker** repository for the back-end code.

**Do not create a new repository!**

# Step 2. Check dependencies

Check your **package.json** file and ensure that you already have the following dependencies installed:

* **bootstrap**
* **lodash**
* **nanoid**

# Step 3. Install additional dependencies

For this assignment, install the following dependencies as well:

* **axios**
* **moment**
* **react-router-dom**
* **react-toastify**

# 

# Step 4. Environment Variables

*Source:* [*https://create-react-app.dev/docs/adding-custom-environment-variables*](https://create-react-app.dev/docs/adding-custom-environment-variables)

Setup an environment variable to configure the API URL.

Create an **.env.local** file:

|  |
| --- |
| VITE\_API\_URL=<http://localhost:5000> or your Google Cloud Custom URL |

Create an **.env.production** file:

|  |
| --- |
| VITE\_API\_URL=Google Cloud Custom URL |

*Replace issue-tracker-student-name with the name of your deployed Issue Tracker API on Google Cloud.*

Update **.gitignore** to check these .env files in:

|  |
| --- |
| # dotenv environment variables file # .env # .env.test  # misc .DS\_Store # .env.local # .env.development.local # .env.test.local # .env.production.local |

# Step 5. Implement components

Follow the directions below to continue implementing the required components.

# index.jsx (5pts)

Import **bootstrap** css and js as shown below: **(1pt)**

|  |
| --- |
| import 'bootstrap/dist/css/bootstrap.min.css'; import 'bootstrap'; |

Import **BrowserRouter** as shown below: **(2pt)**

|  |
| --- |
| import { BrowserRouter } from 'react-router-dom'; |

Wrap **<App>** in a **<BrowserRouter>** as shown below: **(2pt)**

|  |
| --- |
| ReactDOM.render(  <React.StrictMode>  <BrowserRouter>  <App />  </BrowserRouter>  </React.StrictMode>,  document.getElementById('root') ); |

*Instructions continued on the next page.*

# 

# App.jsx (20pts)

Import **react-router-dom** as shown below: **(1pt)**

|  |
| --- |
| import { Routes, Route, Navigate, useNavigate } from 'react-router-dom'; |

Import **react-toastify** as shown below: **(1pt)**

|  |
| --- |
| import { ToastContainer, toast } from 'react-toastify'; import 'react-toastify/dist/ReactToastify.min.css'; |

Add a **<ToastContainer>** to the layout: **(1pt)**

|  |
| --- |
| <div className="App">  <ToastContainer />  <main className="container my-5">  ...  </main> </div> |

Add the following functions to show success and error toast messages: **(2pt)**

|  |
| --- |
| function showError(message) {  toast(message, { type: 'error', position: 'bottom-right' }); } function showSuccess(message) {  toast(message, { type: 'success', position: 'bottom-right' }); } |

Use the **<Routes>** and **<Route>** components to define all of the routes: **(5pt)**

|  |
| --- |
| <main className="container my-5">  <Routes>  <Route path="/" element={<Navigate to="/login" />} />  <Route path="/login" element={<LoginForm />} />  <Route path="/register" element={<RegisterForm />} />  <Route path="/bug/list" element={<BugList />} />  <Route path="/bug/:bugId" element={<BugEditor />} />  <Route path="/user/list" element={<UserList />} />  <Route path="/user/:userId" element={<UserEditor />} />  <Route path="\*" element={<NotFound />} />  </Routes> </main> |

*(Stub out and import the required components, if needed.)*

Update the state variables as shown below: **(3pt)**

* Add an **auth** state variable to store information about the logged in user.
* And add the **navigate** function.

|  |
| --- |
| const [auth, setAuth] = useState(null); const navigate = useNavigate(); |

Add functions to update the state when the user logs in and logs out. **(5pt)**

|  |
| --- |
| function onLogin(auth) {  setAuth(auth);  navigate('/bug/list');  showSuccess('Logged in!'); } function onLogout() {  setAuth(null);  navigate('/login');  showSuccess('Logged out!'); } |

# Navbar.jsx (30pts)

Add a **Navbar** component, if you haven't yet:

|  |
| --- |
| function Navbar() {  return (  <header>  *YOUR NAVBAR HERE*  </header>  ); }  export default Navbar; |

Semantically navbars should have the following HTML structure: **(5pt)**

|  |
| --- |
| <header>  <nav>  <ul>  <li><a>Link Text</a></li>  ... more links here  </ul>  </nav> </header> |

* To support **react-router,** use **NavLink** elements instead of **<a>** elements in the navbar.

|  |
| --- |
| import { NavLink } from 'react-router-dom';  <NavLink className="nav-link" to="/login">Login</NavLink> |

When all the previous steps are completed, add the Navbar component to the layout in **App.js (1pt)**

|  |
| --- |
| <div className="App">  <Navbar />  <ToastContainer />  <main className="container my-5">  ...  </main> </div> |

Pipe the **auth** and **onLogout** variables created previously in **App.js** down the Navbar component. **(1pt)**

|  |
| --- |
| <Navbar auth={auth} onLogout={onLogout} /> |

Receive the passed down properties in **Navbar.js (1pt)**

|  |
| --- |
| function Navbar({ auth, onLogout }) { |

Show appropriate links only when the user is **logged out** in like so: **(5pt)**

|  |
| --- |
| {!auth && (<li>...</li>)} |

Show appropriate links only when the user is **logged in** like so: **(5pt)**

|  |
| --- |
| {auth && (<li>...</li>)} |

Handle the **Logout** link as shown below: **(2pt)**

|  |
| --- |
| function onClickLogout(evt) {  evt.preventDefault();  onLogout(); }  <NavLink className="nav-link" to="/login" onClick={(evt) => onClickLogout(evt)}>  Logout </NavLink> |

# Footer.jsx (5pts)

Create a simple Footer component as shown below: **(3pt)**

|  |
| --- |
| function Footer() {  return (<footer className="bg-dark text-white p-2">  &copy; *Student Name Here 2021*  </footer>) }  export default Footer; |

*Replace Student Name Here 2024 with your own name and the current year.*

And then add the Footer component to the layout in **App.js (2pt)**

|  |
| --- |
| <div className="App">  <ToastContainer />  <main className="container my-5">  ...  </main>  <Footer /> </div> |

*Instructions continued on the next page.*

# LoginForm.jsx (20pts)

This component should provide an email/password field for the user to login.

Follow the directions below to implement this component.

Import all of the following, at the top of the file:

|  |
| --- |
| import { useState } from 'react'; import axios from 'axios'; import \_ from 'lodash';  import { Link } from 'react-router-dom'; |

Pipe the **onLogin** and **showError** variables from the **App** component, to the **LoginForm** component:

|  |
| --- |
| <LoginForm onLogin={onLogin} showError={showError} />  function LoginForm({ onLogin, showError }) { |

Add the following state variables, with the **useState** hook: **(2pt)**

* **email**
* **password**
* **error**
* **success**

*Instructions continued on the next page.*

Wrap the whole form in a **<form>** element. **(2pt)**

Wire each field up to display an error message if it contains invalid data, as shown in the lecture: **(2pt)**

* **email is required, and must contain an @ sign**
* **password is required, and must be at least 8 characters long**

Add a submit button to the bottom of the form, and wire its onClick event to an event handler: **(10pt)**

* If any field contains invalid data, display an error message underneath the submit button and do not send the HTTP request. (Also display an error message as a toast.)
* Otherwise, use [**axios**](https://www.npmjs.com/package/axios) to send a **POST** request to the **/api/user/login** route of our API.
* If the request comes back successfully:
  + Call the **onLogin()** event handler with this **auth** object.
* If the request fails:
  + Display the returned error message, if **err?.response?.data?.error** is a string.
  + Display the returned details, if **err?.response?.data?.error?.details** is not null/undefined.
  + Otherwise, display **err.message**
  + Display an appropriate toast message for all errors.

Also add a **<Link>** to go to the **/register** route, in case the user does not have an account yet. **(2pt)**

# 

# RegisterForm.jsx (20pts)

This component should provide a registration form to the user.

Follow the directions below to implement this component.

Import all of the following, at the top of the file:

|  |
| --- |
| import { useState } from 'react'; import axios from 'axios'; import \_ from 'lodash';  import { Link } from 'react-router-dom'; |

Pipe the **onLogin** and **showError** variables from the **App** component, to the **RegisterForm** component:

|  |
| --- |
| <RegisterForm onLogin={onLogin} showError={showError} />  function RegisterForm({ onLogin, showError }) { |

Add the following state variables, with the **useState** hook: **(2pt)**

* **email**
* **password**
* **passwordConfirm**
* **givenName**
* **familyName**
* **error**
* **success**

*Instructions continued on the next page.*

Wire each field up to display an error message if it contains invalid data, as shown in the lecture: **(2pt)**

* **email is required, and must contain an @ sign**
* **emailConfirm is required, and must match the email field**
* **password is required, and must be at least 8 characters long**
* **passwordConfirm is required, and must match the email field**
* **givenName is required** (and has no other conditions!)
* **familyName is required** (and has no other conditions!)

Add a submit button to the bottom of the form, and wire its onClick event to an event handler: **(10pt)**

* If any field contains invalid data, display an error message underneath the submit button and do not send the HTTP request. (Also display an error message as a toast.)
* Otherwise, use [**axios**](https://www.npmjs.com/package/axios) to send a **POST** request to the **/api/user/register** route of our API.
* If the request comes back successfully:
  + Call the **onLogin()** event handler with this **auth** object.
* If the request fails:
  + Display the returned error message, if **err?.response?.data?.error** is a string.
  + Display the returned details, if **err?.response?.data?.error?.details** is not null/undefined.
  + Otherwise, display **err.message**
  + Display an appropriate toast message for all errors.

Also add a **<Link>** to go to the **/login** route, in case the user already has an account. **(2pt)**

# BugListItem.jsx (20pts)

This component should implement the required code to display a single item within a list.

Accept the item to be displayed via an **item** property:

|  |
| --- |
| function BugListItem({ item }) { |

The following Bootstrap classes must be used to style the item:

* **card** or **table row** to display the item (1pt)
* **mb-2** or **mb-3** to separate adjacent cards in the list (1pt)
* Do not include a **card-header** (1pt)
* Display the following inside the **card-body** (1pt)
  + Display the bug's title (with class **card-title**) (2pt)
  + Display who the bug is assigned to (2pt)
  + Display the classification as a badge:
    - **badge** **bg-success,** when classification=approved (1pt)
    - **badge** **bg-danger,** when classification=unapproved (1pt)
    - **badge** **bg-danger,** when classification=duplicate (1pt)
    - **badge bg-warning,** when classification=unclassified (1pt)
  + Display the open/closed status as a badge:
    - **badge** **bg-success,** when open (2pt)
    - **badge** **bg-danger,** when closed (2pt)
  + Additional relevant information may be added.
  + Do not include the full bug description or steps to reproduce.
* Display the following inside the **card-footer**
  + When the bug was created and by who. (2pt)
  + Use [**moment.js**](https://momentjs.com/) to format the date (2pt) [for example **moment(item.createdOn).fromNow()**]

# BugList.jsx (20pts)

This component should implement the required code to display a list of bugs. It should also include the required code to retrieve this list from the API.

*(It does not need to include the search interface at this time.)*

Accept the following properties:

|  |
| --- |
| function BugList({ auth, showError, showSuccess }) { |

Use [**axios**](https://www.npmjs.com/package/axios) to make a **GET** request to the **/api/bug/list** route. **(5pt)**

* Show a spinner while the request is pending. **(3pt)**
* Show an error message if the list fails to load. *(Show it in the layout and as a toast.)* **(3pt)**
* Once the list is retrieved without error, if there are no items in the list display the message: **(3pt)** **"No bugs found."**
* If bugs are retrieved, then display each bug using the **<BugListItem />** component created above. **(3pt)**
* Add a **<Link>** to the BugListItem, that links to the **/bug/:bugId** page. **(3pt)**

*Instructions continued on the next page.*

# BugEditor.jsx (30pts)

This component should implement the interface for editing a bug. It should also include the required code to retrieve the bug from the API and send updates to the API.

Accept the following properties:

|  |
| --- |
| function BugEditor({ auth, showError, showSuccess }) { |

Wrap the whole form in a **<form>** element.

Use [**axios**](https://www.npmjs.com/package/axios) to make a **GET** request to the **/api/bug/:bugId** route. **(5pt)**

* Show an error message if the request fails, and do not show the form. **(3pt)**  *(Show the message in the layout and as a toast.)*
* Once the bug is retrieved, display the form. **(1pt)**
* Store each of the bug in a state variable: **(3pt)**

When the user clicks the submit button, use [**axios**](https://www.npmjs.com/package/axios) to make a **PUT** request to the **/api/bug/:bugId** route. **(10pt)**

* Show an error message if the request fails. **(3pt)** *(Show the message in the layout and as a toast.)*
* Show a success message if the request succeeds. **(3pt)** *(Show the message in the layout and as a toast.)*

# UserListItem.jsx (20pts)

This component should implement the required code to display a single item within a list.

Accept the item to be displayed via an **item** property:

|  |
| --- |
| function UserListItem({ item }) { |

The following Bootstrap classes must be used to style the item:

* **card** to display the item as a Bootstrap card (1pt)
* **mb-2** or **mb-3** to separate adjacent cards in the list (1pt)
* Do not include a **card-header** (1pt)
* Display the following inside the **card-body** (1pt)
  + Display the user's full name (with class **card-title**) (2pt)
  + Display the user's email address (2pt)
  + Display the user's roles as a badges: (8pt)
    - **badge** **bg-primary,** for any role
    - **badge** **bg-danger,** for users than have no role
    - Handle users that do not have an assigned role.
    - Handle users that have multiple roles
  + Do not include the user's password.
* Display the following inside the **card-footer**
  + When the user was registered. (2pt)
  + Use [**moment.js**](https://momentjs.com/) to format the date (2pt) [for example **moment(item.createdOn).fromNow()**]

# UserList.jsx (20pts)

This component should implement the required code to display a list of users. It should also include the required code to retrieve this list from the API.

*(It does not need to include the search interface at this time.)*

Accept the following properties:

|  |
| --- |
| function UserList({ auth, showError, showSuccess }) { |

Use [**axios**](https://www.npmjs.com/package/axios) to make a **GET** request to the **/api/user/list** route. **(5pt)**

* Show a spinner while the request is pending. **(3pt)**
* Show an error message if the list fails to load. *(Show it in the layout and as a toast.)* **(3pt)**
* Once the list is retrieved without error, if there are no items in the list display the message: **(3pt)** **"No users found."**
* If users are retrieved, then display each user using the **<UserListItem />** component created above. **(3pt)**
* Add a **<Link>** to the UserListItem, that links to the **/user/:userId** page. **(3pt)**

*Instructions continued on the next page.*

# UserEditor.jsx (30pts)

This component should implement the interface for editing a user. It should also include the required code to retrieve the user from the API and send updates to the API.

Accept the following properties:

|  |
| --- |
| function UserEditor({ auth, showError, showSuccess }) { |

Wrap the whole form in a **<form>** element.

Use [**axios**](https://www.npmjs.com/package/axios) to make a **GET** request to the **/api/user/:userId** route. **(5pt)**

* Show a spinner while the request is pending. **(1pt)**
* Show an error message if the request fails, and do not show the form. **(3pt)**  *(Show the message in the layout and as a toast.)*
* Once the bug is retrieved, display the form. **(1pt)**
* Store each of the following user properties in a state variable: **(3pt)**
  + **email**
  + **password** (not returned by API)
  + **givenName**
  + **familyName**
  + **fullName** (must be a separate field on the form!)
  + **role**

When the user clicks the submit button, use [**axios**](https://www.npmjs.com/package/axios) to make a **PUT** request to the **/api/user/:userId** route. **(10pt)**

* Show an error message if the request fails. **(3pt)** *(Show the message in the layout and as a toast.)*
* Show a success message if the request succeeds. **(3pt)** *(Show the message in the layout and as a toast.)*