CMPS 356 – Fall 2022
Web Applications Design and Development

Lab 05
React Hooks and Routing

Objective

1. Using outlet contexts (`useOutletContext`)
2. Navigating programmatically (`useNavigate`)
3. Setting protected routes
4. Using multiple layouts with layout routes
5. Memoizing values and functions using callbacks (`useMemo` and `useCallback`)
6. Prioritizing updates using transitions and deferred values (`useTransition` and `useDeferredValue`)

Prerequisites

1. React Router tutorial: https://reactrouter.com/en/main/getting-started/tutorial
2. React hooks API reference: https://reactjs.org/docs/hooks-reference.html
3. Debouncing and throttling: https://css-tricks.com/debouncing-throttling-explained-examples

1. Outlet Contexts and Programmatic Navigation

1. Create a new directory `01-routing-navigation` and reuse the code of the last exercise from the previous lab under it.
2. Fetch the list of country facts and store it in the `App` component.

```
const App = () ⇒ {
  const [facts, setFacts] = useState(null);
  useEffect(() ⇒ {
    fetchFacts()
    ...
  }, []);
```

3. Rename your `Gallery` page to `Facts` and display the list of countries in a dropdown list instead of using multiple links.
4. Navigate to the corresponding country and display its facts when the selection changes. Add an extra default option for the index page.
5. Navigate to the `Facts` page as fallback when the country provided is not part of the list. Use the path parameter to check whether the country code is valid or not.
6. Use an outlet context to pass the list of country facts to the `Country` component to avoid the redundant requests used to fetch the facts for each country.
7. How can the country selection be made more efficient to the end-user?

2. Protected Routes and Multiple Layouts

1. Create a new directory `02-protected-routes` and reuse the code of the previous exercise.
2. Add a `Login` page that allows the end-user to authenticate by providing their username and email address. Store the user state and move your router and routes to the `App` component.
3. Use a different layout for the `Login` page that does not include a header nor a footer. This requires regrouping your routes using layout routes and creating a `Basic` component for the new layout.

   ```
   <Route element={<Layout />}>
     ...
   </Route>
   <Route element={<Basic />}>
     ...
   </Route>
   ```

4. Display the username in the header along with a button to unauthorize (logout). The end-user should be automatically redirected the landing page after logging out.
5. Create a new `Photos` page that is only accessible after the user has authenticated. This requires a new route entry and component, `ProtectedRoute`, to handle such protected pages (also called private pages). Update your navigation bar to link to this new page.
6. Create a protected `Profile` page that allows the end-user to update their username or email address. Add the corresponding route entry and navigation bar link.
7. Update your login/logout button such that the end-user can manually authenticate by clicking it.
8. Hide the protected links from the navigation bar when the end-user is not authenticated.

3. Value Memoization and Update Deferral

1. Create a new directory `03-memoization-deferral` and an application under it.
2. Create an `Input` component that holds a text value linked to a state that can be set by the end-user through a textbox.
3. Update your component with a method to return a value passed as a property to the component after busy waiting a certain duration in milliseconds. The returned value is used to update a state variable. Use the following function to simulate a time-consuming computation:

   ```
   const wait = (value, duration) => {
     const date = Date.now();
     let current = null;
     do {
       current = Date.now();
     } while (current - date < duration);
     return value;
   };
   ```

4. Try changing the text input quickly and observe the responsiveness of the overall user interface when dealing with increasingly longer waiting duration.

5. Use a hook to memoize the return value of your waiting method so that it is reused whenever the component is rendered.
6. Add a slider that, whenever changed, returns a given value asynchronously after a certain duration in milliseconds; we are trying to simulate an asynchronous request. The returned value is used to update a state variable. Use the following function to simulate a request that resolves with a given value after a certain given duration:

   ```
   const sleep = (value, duration) => {
     return new Promise((resolve) =>
       setTimeout(() => {
         resolve(value);
       }, Math.random() * duration)
     );
   };
   ```
7. Try changing the slider value quickly. How responsive and linear is the experience with increasingly longer waiting durations?
8. Use a hook to defer the state associated with the value of the slider. Is that sufficient to enhance the overall feel of the experience?
9. Memoize the call using a deferred value of the state variable instead of the state variable itself.
10. How can the end-user be kept updated about the status of the simulated request? Use a transition hook.