CMPS 356 – Fall 2022
Web Applications Design and Development

Lab 04
React Context and Routing

Objective

1. Creating, providing, and consuming contexts for managing state,
2. Exploring layouts when designing and composing components,
3. Configuring and routing between different pages/components,
4. Using active links, outlets, and path hooks (`useParams`).

Prerequisites

1. React Context tutorial: https://reactjs.org/docs/context.html
2. React Router tutorial: https://reactrouter.com/en/main/getting-started/tutorial

1. Providing and Consuming Context

1. Create a new directory `01-context` and copy the last exercise from the previous lab under it.
2. How can we use these different components separately without having to include them part of a `Gamer` instance? Note that the state can be lifted all the way up to the entry point of our application.
3. Create a `GamerContext` context provider and use it for storing, accessing, and manipulating the state of the username, age, status, and avatar of the gamer in all your components.
4. Reduce the state of your context provider to have more granular control over the state updates.

2. Layouts using Components

1. Create a new directory `02-layout` and a React application under it.
2. Create a `jsconfig.json` file at the root of your project with the following content (this will facilitate importing files using only their absolute paths relative to `src`):

```
{
  "compilerOptions": {
    "baseUrl": "src"
  },
  "include": ["src"]
}
```

3. Create a `Layout` component under `src/layouts`.
4. Render the children passed to the `Layout` component under a `<main>` tag.
5. Create `Header`, `Navigation`, and `Footer` components under `src/components` and update your `Layout` component to include them.

```
const Layout = ({ children }) => {
  return (
    <>
```

```
      <Header />
      <main>{children}</main>
      <Footer />
    </>
  );
};
```

6. Add sample content under a `Layout` instance and verify that the layout, the various components, and the sample content are all being rendered correctly.

```
const App = () => {
  return (
    <div className="App">
      <Layout>
        <p>Lorem ipsum</p>
      </Layout>
    </div>
  );
};
```

3. Path Routing and Navigation Links

1. Create a new directory `03-routing` and reuse the source from the previous exercise.
2. Create a couple of pages: `Home`, `Gallery`, and `About` components under `src/pages`. Use the same layout for every page as the App component. How can we navigate between these pages?
3. Add the React Router library to your project: `npm install react-router-dom@6`.
4. Update your entry point `index.js` with a route configuration for the various paths and corresponding pages.

```
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<App />} />
        <Route path="gallery" element={<Gallery />} />
        <Route path="about" element={<About />} />
      </Routes>
    </BrowserRouter>
  </React.StrictMode>
);
```

5. Update your navigation component with links to the various pages. How can we use a unified layout for all pages?

```
const Navigation = (props) => {
  return (
    <>
      <nav>
        <Link to="/">Home</Link>
        <Link to="/gallery">Gallery</Link>
        <Link to="/about">About</Link>
      </nav>
    </>
  );
};
```

6. Nest your routes in your entry point routing configuration then create an outlet in your `App` component so that you can navigate between the different pages by swapping them in and out of this outlet. You will notice that the layout elements are repeated twice on every page.

```
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<App />}>
          <Route path="gallery" element={<Gallery />} />
          <Route path="about" element={<About />} />
        </Route>
      </Routes>
    </BrowserRouter>
  </React.StrictMode>
);
```

7. Update your components so that the layout is defined only once in your `App` component.
8. Use an index route to render `Home` as the default page for `/`.

```
<Route path="/" element={<App />}>
  <Route index element={<Home />} />
  ...
```

9. Add a default (no match) route to your configuration to handle non-existing paths.

```
<Route path="*" element={ <Layout><p>404</p></Layout> } />
```

10. Use active links in your `Navigation` component to highlight the current page.
11. Fetch the list of country facts in your `Gallery` component from `https://restcountries.com/v3.1/all` and create a link to every country using their CCA3 code as the path. You will notice that these links lead to nowhere and will be handled by the no-match rule.
12. Display the country code of the corresponding country by defining a route to match the nested paths under `/gallery`. You will need to add an outlet to `Gallery` and create a `Country` component that renders the country code.

```
<Route path="gallery" element={<Gallery />}>
  <Route path=":cca3" element={<Country />} />
</Route>
```

13. Use active links in your `Gallery` component to highlight the currently active country.
14. What if we wanted to display the flag of the corresponding country or any other fact? Update your `Country` component and use `https://restcountries.com/v3.1/alpha/{code}` to fetch the facts for the matched country and display its name and flag. The CCA3 path parameter can be retrieved using a path hook, `useParams`, inside your `Country` component.

15. As you navigate through the pages, you will notice that the list of countries is fetched every single time we visit `/gallery` since it is being mounted/unmounted. The facts for a given country are also fetched every single time. How can we fetch the list of facts for all countries once and reuse it throughout? How can this list of facts be shared among countries without having to fetch the facts every single time we navigate to a different country?