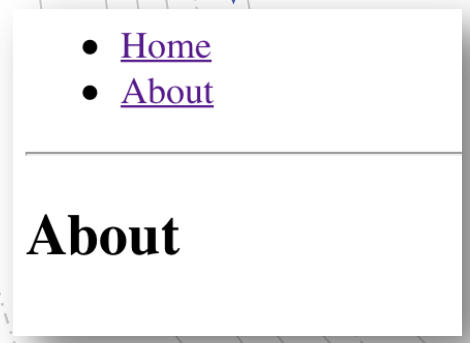
Lab 5
React Router

CSCI2720 Building Web Applications

Agenda

- Basics of React Router
- Custom links
- URL parameters
- No-match route
- Functional vs class components

Installing React-router



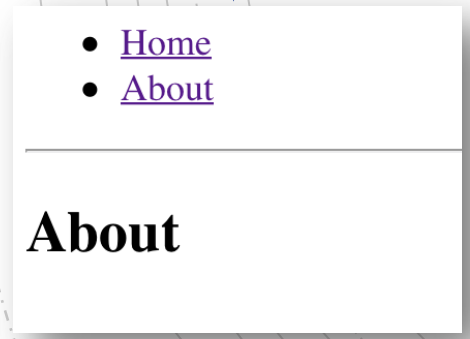
- Similar to Lab 4, you are working on plain HTML/JS files
  - React, ReactDOM, and ReactRouterDOM will be loaded from CDN
- Start with this zip file

  http://www.cse.cuhk.edu.hk/~chuckjee/2720lab5/lab5.zip
  - Necessary libraries are included
  - The current contents are just like in the example in lecture slides
- Load this folder in ***Web Server for Chrome*** and visit the page in Chrome

Basic Components

- Routers
  - `<BrowserRouter>` (renamed as **`<Router>`** in the given file)
    - For modern browsers, supporting HTML5 History API with states, e.g. the *Back* button
- Route matchers
  - **`<Switch>`** looks at children `<Route>` elements for the first match, and ignore others
  - **`<Route>`** matches URL against the **`path="…"`** attribute
    - If **`exact`** is specified, the whole URL is matched
    - Otherwise, `path="/"` matches everything since all URL starts with /
- Route changers
  - **`<Link>`** allows specifying the **`to`** attribute
- *See: https://reactrouter.com/web/guides/primary-components*

In a nutshell…



- The given files has these features:
  - A list of links, and the linked component are displayed inside the component **App**
    - If the link ***Home*** is visited (URL becomes `/`), the **Home** component is shown
    - If the link ***About*** is visited (URL becomes `/about`), the **About** component is shown instead

Task 1:

Custom links

- We can introduce some appearance difference to the *active* link in the list

1. Inside **App**, change the link list to

```
<ul>
  <LongLink
    activeOnlyWhenExact={true}
    to="/"
    label="Home"
  />
  <LongLink to="/about" label="About" />
</ul>
```

2. Add this line to the beginning of `app.jsx`

```
const {useRouteMatch, useParams, useLocation} = ReactRouterDOM;
```

   - We are only using **`useRouteMatch`** in this task, but the other two will be used later

Task 1:

Custom links

3. Set up a new component **LongLink**

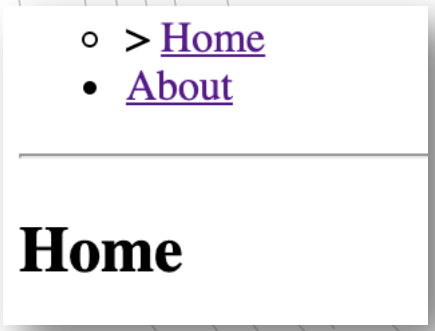
```
function LongLink({label, to, activeOnlyWhenExact}) {
  let match = useRouteMatch({
    path: to,
    exact: activeOnlyWhenExact
  });
  return (
    <li className={match ? "active" : ""}>
      {match && "> "}
      <Link to={to}>{label}</Link>
    </li>
  );
}
```

- The argument list **`LongLink({label, …})`** shows a shorthand of loading the props
  - It is like setting **`let label = props.label;`**

- Note that this is a *functional component*, to make use of the "hook" **`useRouteMatch()`**
  - The **`match`** object has the information of routing path and whether an exact match is needed

Task 1:

Custom links



- If you have set up **`LongLink`** successfully, when you reload the page in Chrome, you can see that there is a **>** in front of the active link
- The list bullet is also different. *How* was this done?
- *Example was adapted from https://reactrouter.com/web/example/custom-link*

Task 2:

URL parameters

- A variable could be matched inside the URL

1. Set up three more **<LongLink>**, pointing to **/file/fileA**, **/file/fileB**, and **/file/fileC**
   - You can decide what *labels* they should take
2. Under **<Switch>**, add one more **<Route>**

```
<Route path="/file/:id" component={File} />
```

3. Set up a new component **File**

```
function File() {
  let { id } = useParams();
  return (
    <div>
      <h3>ID: {id}</h3>
    </div>
  );
}
```

   - Where does **id** come from?

Task 2:

URL parameters

- Using the parameter `:id`, the string could be automatically captured for use with the `useParams()` hook
- This is especially useful for ***pattern matching*** in URL
- *Example is adapted from https://reactrouter.com/web/example/url-params*
- *Read more here: https://medium.com/better-programming/using-url-parameters-and-query-strings-with-react-router-fffdcea7a8e9*

Task 3:

No-match route

- Traditionally, a web server would return status **404** with an error page to a URL not found on the server
- We can also do it here

1. Add a wrong to URL with **LongLink**
2. Add a new **Route** at the end (*why?*) of the list

```
<Route path="*" component={NoMatch} />
```

3. Set up a new **NoMatch** component

```
function NoMatch() {
  let location = useLocation();
  return (
    <div>
      <h3>
  No match for <code>{location.pathname}</code>
      </h3>
    </div>
  );
}
```

   - The **useLocation()** hook tells us what URL was bringing to this page

- *Example is adapted from* *https://reactrouter.com/web/example/no-match*

Functional vs class components

- Our "original" components were written in classes
- New components in the lab today are written in functions
- What is better?
  - ***Classes***: more traditional way to understand objects, clear use of props/states
  - ***Functions***: cleaner code, shifting to the use of hooks
- *Lots of tutorials* on both of the two
- Either is fine, or even a mix of both
- Learn more about hooks of React Router: https://css-tricks.com/the-hooks-of-react-router/

Still a long way to go…

- You have made a very simple SPA with dynamic routing
  - Ideally, all requests towards your app should be handled by `index.html` for routing to be making sense
  - Forwarding ***all requests*** to `index.html` requires adjustments in the web server… which is not easy for Web Server for Chrome
- This lab mainly serves as an exercise for you to *know better about React*
- You shall explore further yourself!

Submission

- No submission is needed for labs
- What you have done could be useful for your further exploration or the upcoming assignment
- **Please keep your own file safely**