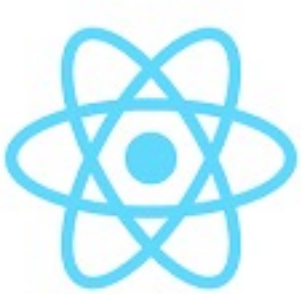


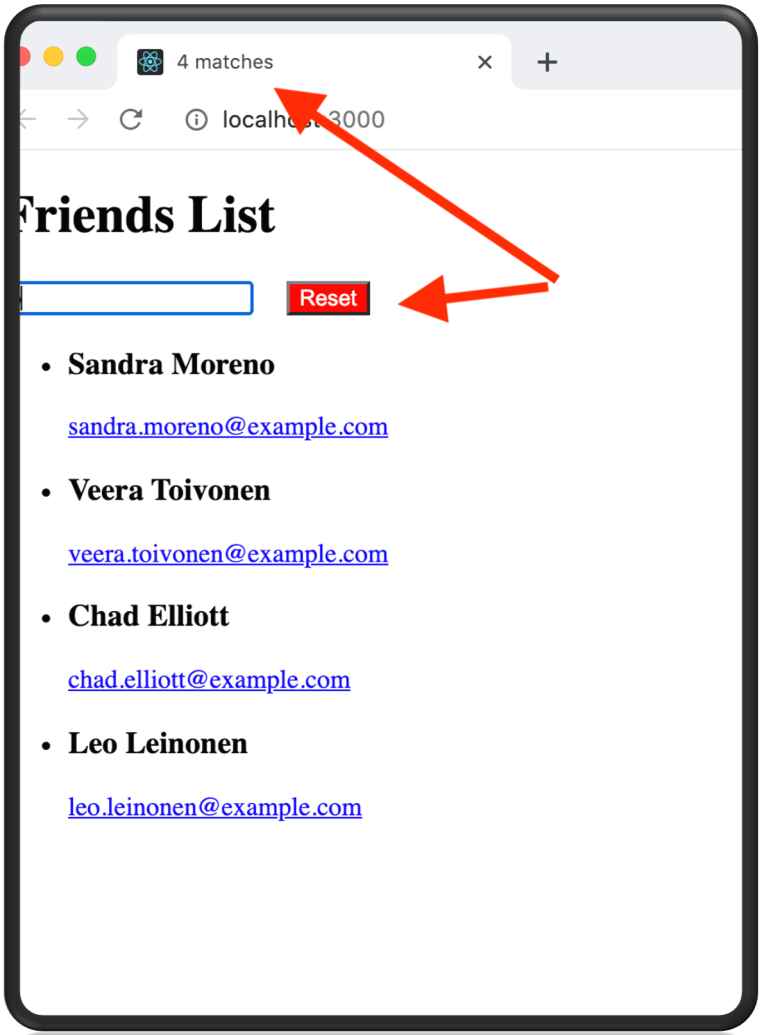
ReactJS.

The Component model (Contd)

Topics

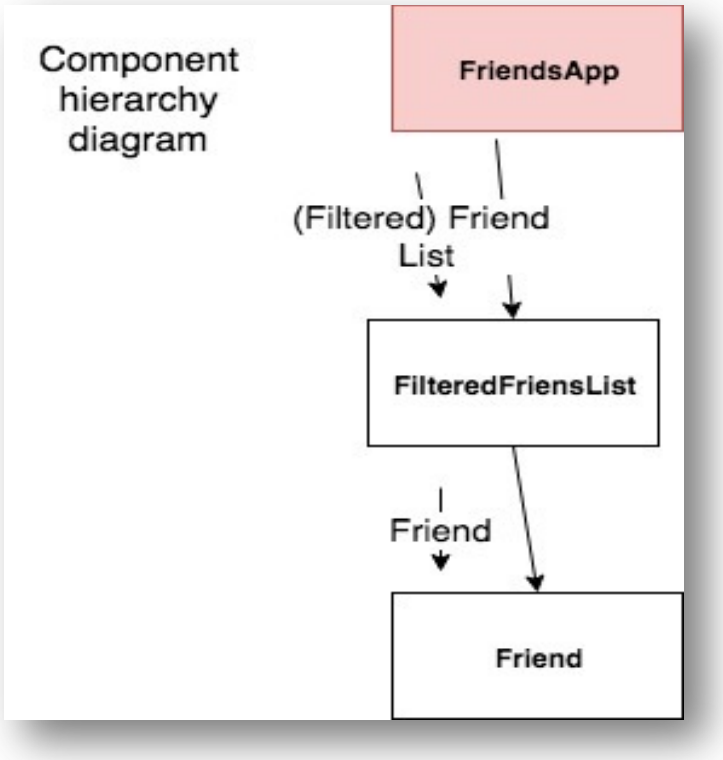
- **Hooks and Component Lifecycle.**
- **Data Flow patterns – Data Down, Action Up pattern.** ✔

Sample App 1 – Version 2



- **App UI changes:**
  1. **A 'Reset' button – loads a new list of friends. Overwriting the current list.**
  2. **Browser tab title - shows # of matching friends (side effect).**
- **See lecture archive for source code**

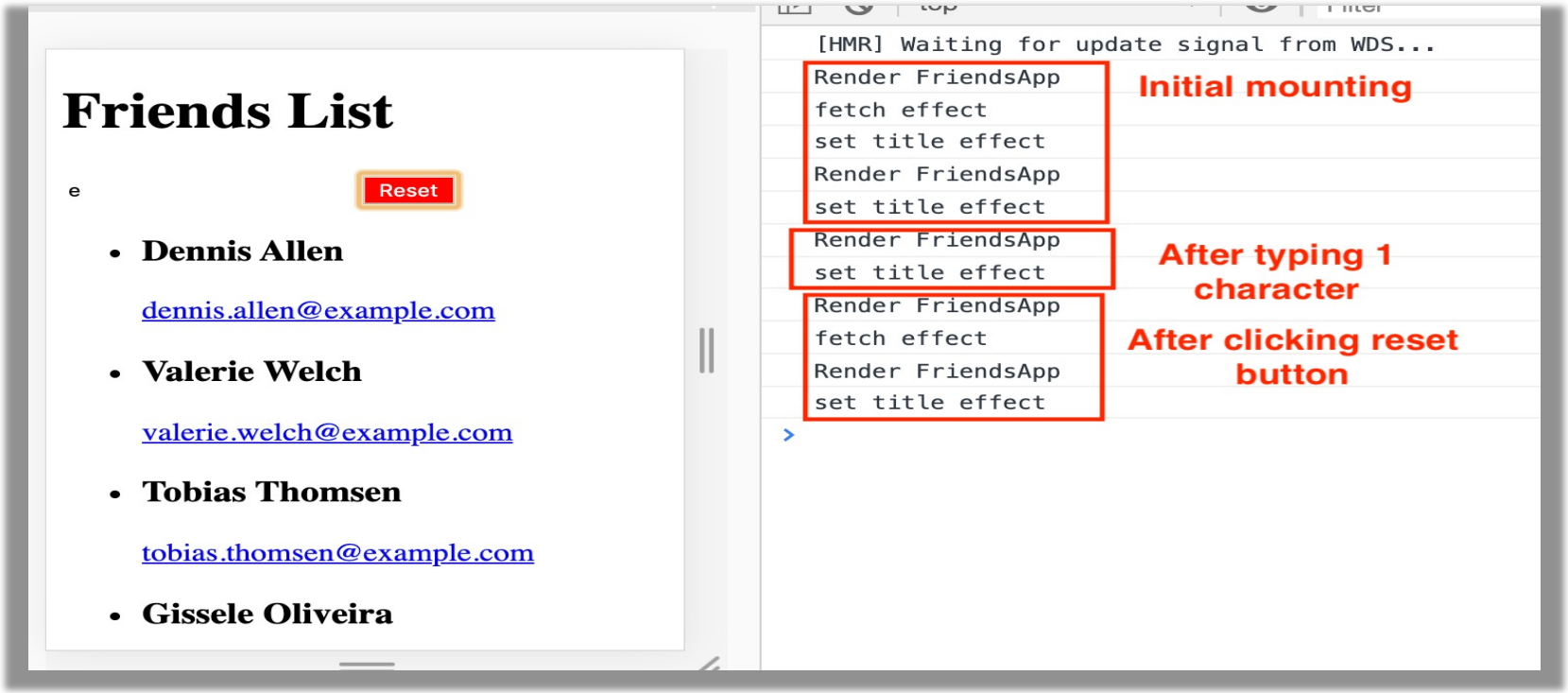
Sample App 1 (v2) - Design



- **3 state variables:**
  1. **List of friends from API.**
  2. **Text box content.**
  3. *Reset button toggle.*

- **2 side effects:**
  1. **'Fetch API data' - dependent on change to** reset button **toggle.**
  2. **'Set browser tab title' - dependent on change to matching list length.**

Sample App 1 (v2) - Events



Sample App 1 - Events.

- **On mounting of** FriensApp **component:**
  Both effects execute (Set browser tab to ’0 matches’).
  → **‘Fetch data’ effect changes ‘friends list’ state.**
  → **Component re-renders + ‘Set browser tab’ effect executes.**
- **On typing a character in the text box:**
  ‘Text box’ state change.
  → FriendsApp **rerenders + Matching friends list length changes**
  → **‘Set browser title’ effect executes.**
- **On clicking Reset button:**
  ‘Reset toggle’ state changes.
  → FriendsApp **re-renders.**
  → **‘Fetch data’ effect executes.**
  → **‘Friends list’ state changes.**
  → **FriendsApp re-renders + Matching list length changes.**
  → **‘Set browser title’ effect executes.**

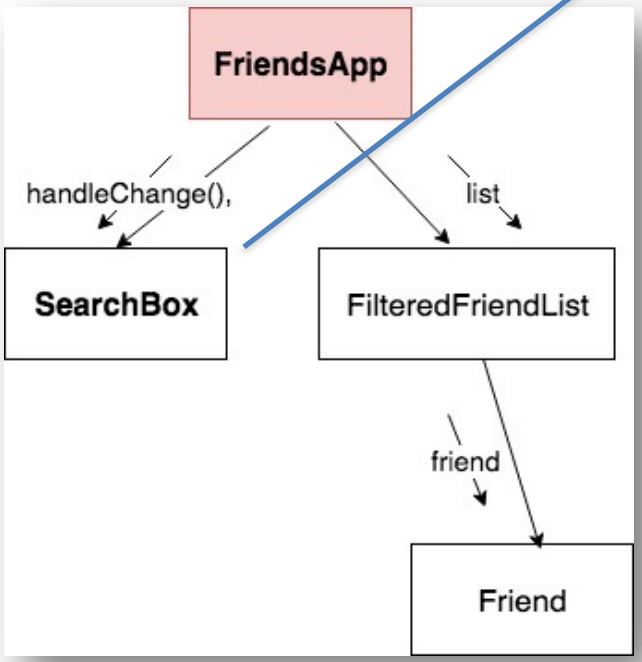
Topics

- **Hooks and Component Lifecycle.** ✔
- **Data Flow patterns – Data Down, Action Up pattern.** ✔

Sample App 2

(Data down, actions up pattern or Inverse data flow pattern)

- **What if a component's** state **is influenced by an** event **in a subordinate component?**
- **Solution: The** data down, action up pattern.



Data down, Action up.

Pattern:

1. Stateful **component** (FriendsApp) **provides a callback to the subordinate** (SearchBox)**.**
2. Subordinate **invokes callback when the event** (onChange) **occurs.**

```
const FriendsApp = () => {
  const [searchText, setSearchText] = useState("");
  const [friends, setFriends] = useState([]);

  useEffect(() => {…
  }, []);

  const filterChange = text =>
    setSearchText(text.toLowerCase());

  const updatedList = friends.filter(friend => {…
  });
  return (
    <>
      <h1>Friends List</h1>
      <SearchBox handleChange={filterChange } />
      <FilteredFriendList list={updatedList} />
    </>
```

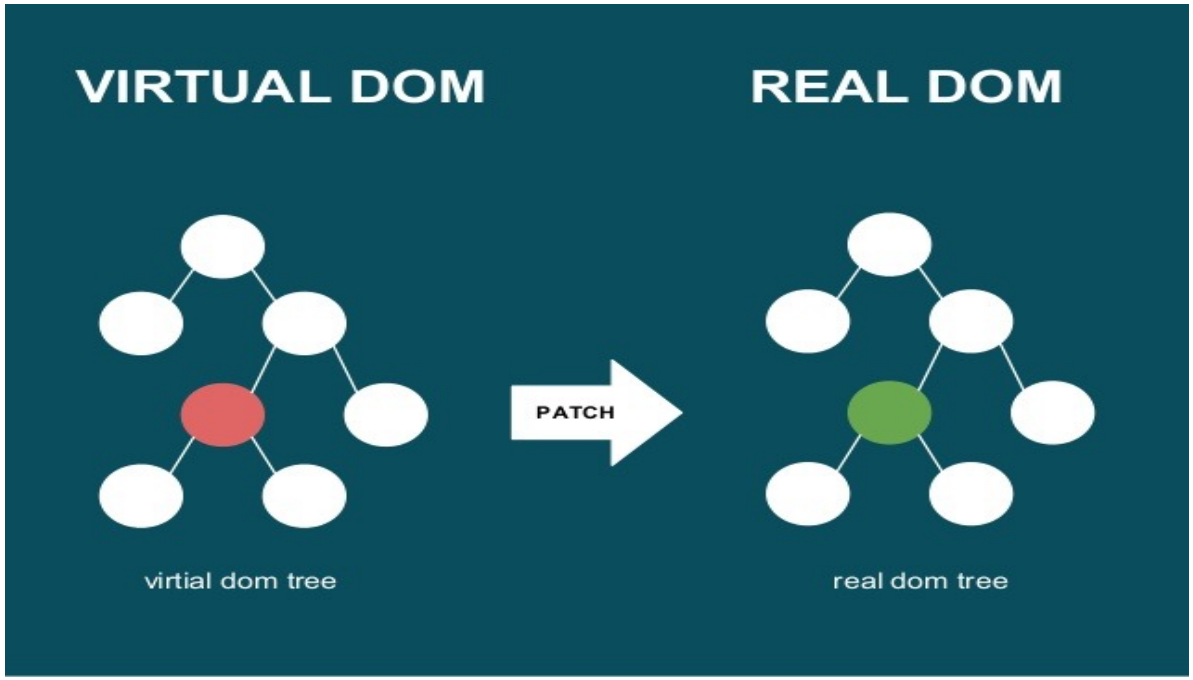
```
const SearchBox = props => {
  const onChange = event => {
    event.preventDefault();
    const newText = event.target.value.toLowerCase();
    props.handleChange(newText);
  };

  return <input type="text" placeholder="Search"
           onChange={onChange} />;
};
```

Summary

- **A state variable change always causes a component to re-render.**
  - **State change logic is usually part of an event handler function.**
  - **Event hadler may be in a subordinate component.**

- **Side effects:**
  - **Always execute at mount time.**
  - **The dependency array will either reference a state variable, a value computed from a state variable, or a prop.**
    - **Can be multiple entries**
  - **Callback performs the side-effect, and may also cause a state change.**

- **Data flows downward, actions flow upward.**



React internals.

Modifying the DOM

- **DOM – an internal data structure representing the browser's current 'display area' ; DOM always** in sync **with the display.**

- **Traditional performance best practice:**
  1. **Minimize access to the DOM.**
  2. **Avoid expensive DOM operations.**
  3. **Update elements offline, then reinsert into the DOM.**
  4. **Avoid changing layouts in Javascript.**
  5. **. . . etc.**

- **Should the developer be responsible for low-level DOM optimization? Probably not.**
  - **React provides a *Virtual DOM* to shield developer from these concerns.**

The Virtual DOM

- **How React works:**
  1. **It create a lightweight, efficient form of the DOM – the Virtual DOM.**
  2. **Your app changes the V. DOM via components' JSX.**
  3. **React engine:**
     1. **Perform *diff* operation between current and previous V. DOM state.**
     2. **Compute the set of changes to apply to real DOM.**
     3. **Batch update the real DOM.**

- **Benefits:**
  a) **Cleaner, more descriptive programming model.**
  b) **Optimized DOM updates and reflows.**

Unidirectional data flow & **Re-rendering**

(revised from previous lecture)

- **What happens when the user types in the text box?**

  User types a character in text box

  → *onChange* ***event handler executes***

  → ***Handler changes a state variable***

  → ***React re-renders*** *FriendsApp component*

  → ***React re-renders children (****FilteredFriendList****) with new prop values.***

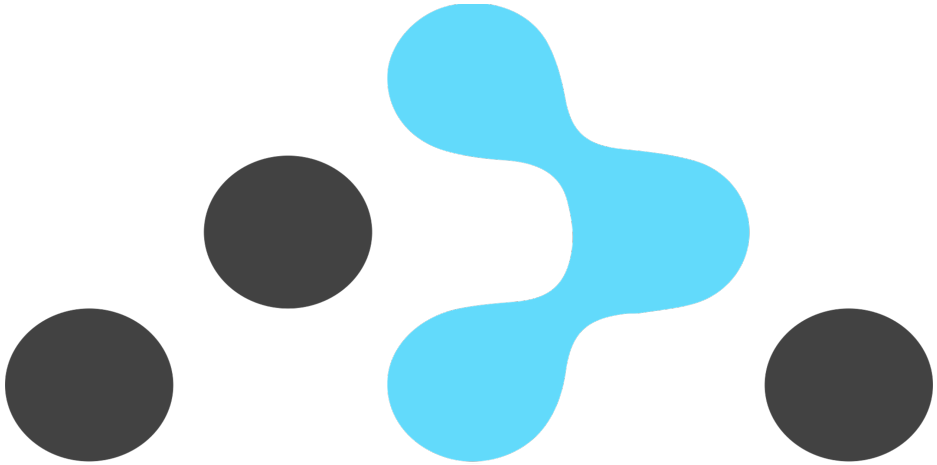
  → ***React re-renders children of*** *FilteredFriendList****.***

  → ***Some components may have unmounted, new ones (re)mounted.***

  → ***(Pre-commit) React diffs the changes between the current and previous Virtual DOM***

  → ***(Commit) React batch updates the real DOM.***

  → ***Browser repaints screen***



Navigation

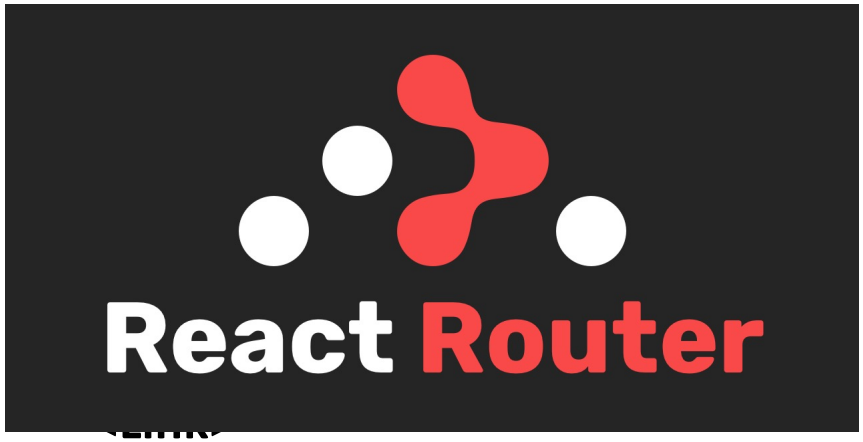
The React Router

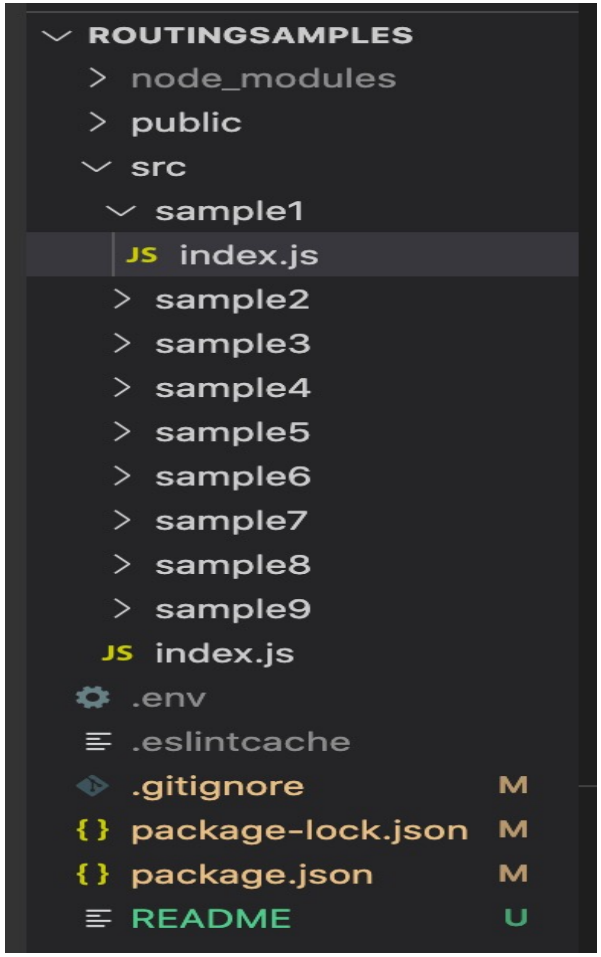
Routing - Introduction

- **Allows multiple views / pages in an app.**
- **Keeps the URL in sync with the UI.**
- **Adheres to traditional web principles:**
  1. **Addressability.**
  2. **Information sharing.**
  3. **Deep linking.**
  - **1st generation AJAX apps violated these principles.**
- **Not supported by the React framework.**
  - **A separate library is required: React Router.**

Demos

- **See the archive.**
- **Each sample demos a routing feature**





Basic routing configuration

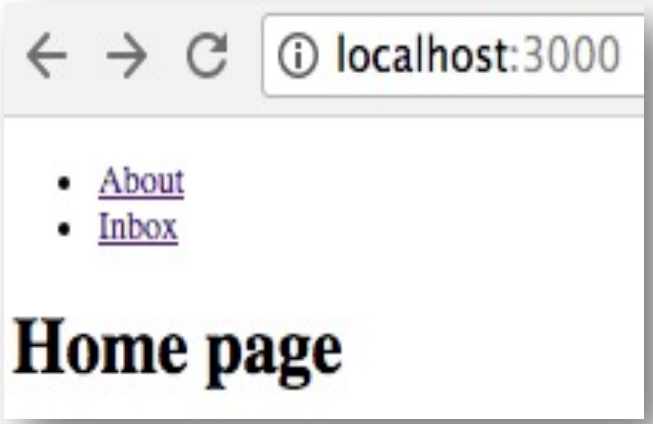
	URL	Components
1	/	Home
2	/about	About
3	/inbox	Inbox

```
17  const App = () => {
18    return (
19      <BrowserRouter>
20        <Routes>
21          <Route path="/about" element={<About />} />
22          <Route path="/inbox" element={<Inbox />} />
23          <Route index element={<Home />} />
24          <Route path="*" element={<Navigate to="/" replace />} />
25        </Routes>
26      </BrowserRouter>
27    );
28  };
```

- **Declarative routing.**
- <BrowserRouter> **- matches browser's URL with a <Route> path.**
- <Route> **-** element **states what's mounted on DOM when match occurs.**
  - element **can take any arbitrary JSX.**
  - **Use** index **for root path case (/).**
  - **Use * path for 404 case.**
  - <Navigate> **changes browser's URL address.**
- ReactDOM.render**() passed an app's Router component.**
- **Ref. src/sample1**

Hyperlinks

- **Use the <Link> component for internal links.**
  - **Use anchor tag for external links - <a href . . . . . >**
- **Ref. src/sample2/**




```
6   const Home = () => {
7     return (
8       <>
9         <ul>
10          <li>
11            <Link to="/about">About</Link>
12          </li>
13          <li>
14            <Link to="/inbox">Inbox</Link>
15          </li>
16        </ul>
17        <h1>Home page</h1>
18      </>
19    );
20  };
```

- **<Link> changes browser's URL address (event)**

  → React Router handles event by consulting its routing configuration

  → Elements mounting/unmounting on/from DOM → Browser updates screen

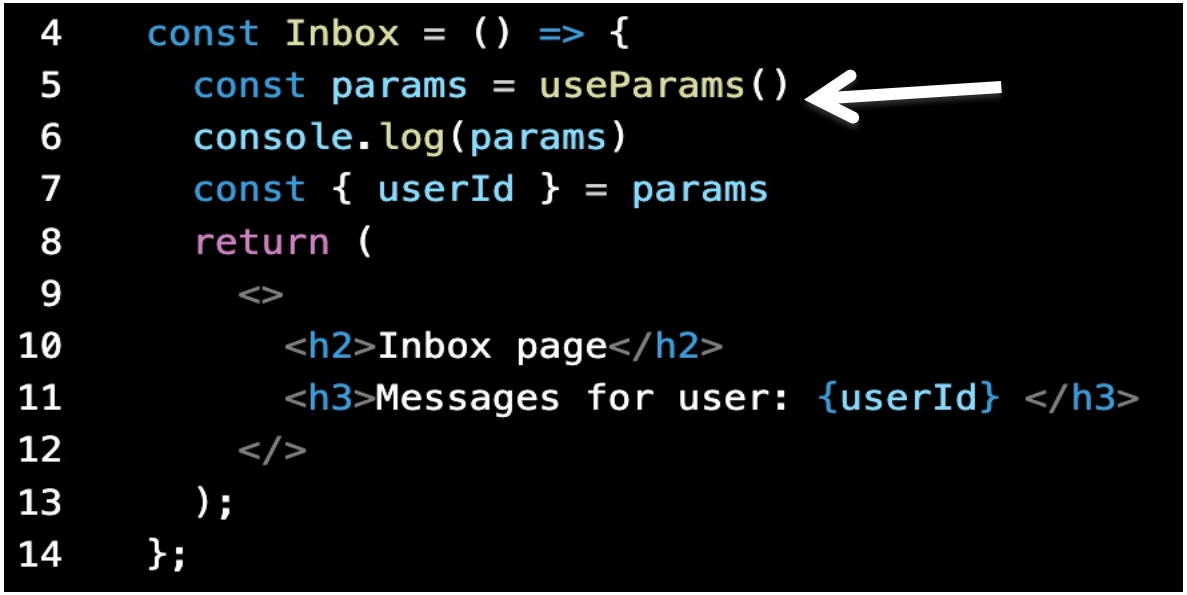
Dynamic segments.

- **Parameterized URLs, e.g. /users/*22*, /users/*12*/purchases**
  - **How do we declare a parameterized path in the routing configuration?**
  - **How does a component access the parameter value?**
- **Ex: Suppose the Inbox component shows messages for a specific user, where the user's id is part of the browser URL**

  e.g /inbox/123 where 123 is the user's id.

- **Solution:** <Route path='/inbox/:userId' element={ <Inbox/> } />
  - **The colon (:) prefixes a parameter in the path; Parameter name is arbitrary.**
  - **Ref src/sample3**

Dynamic segments.



```
4   const Inbox = () => {
5     const params = useParams()
6     console.log(params)
7     const { userId } = params
8     return (
9       <>
10        <h2>Inbox page</h2>
11        <h3>Messages for user: {userId} </h3>
12      </>
13    );
14  };
```

- useParams **hook is provided by React Router library.**
  - **Descructure its returned object to ccess parameter value/**
  - Other useful hooks also provided (see later)
- **More than one parameter allowed.**

  e.g. /users/:userId/categories/:categoryName

Nested Routes

- Objective**: A component's child is dynamically determined from the browser's URL (Addressability).**

- **EX.: (See src/sample4) Given the route:**

  <Route path='/inbox/:userId' element={ <Inbox /> } />,

  use the following rules to determine a nested component hierarchy:

  /inbox/XXX/statistics

  <Inbox>

  <Stats/>

  </Inbox>

  /inbox/XXX/draft

  <Inbox>

  <Drafts/>

  </Inbox>

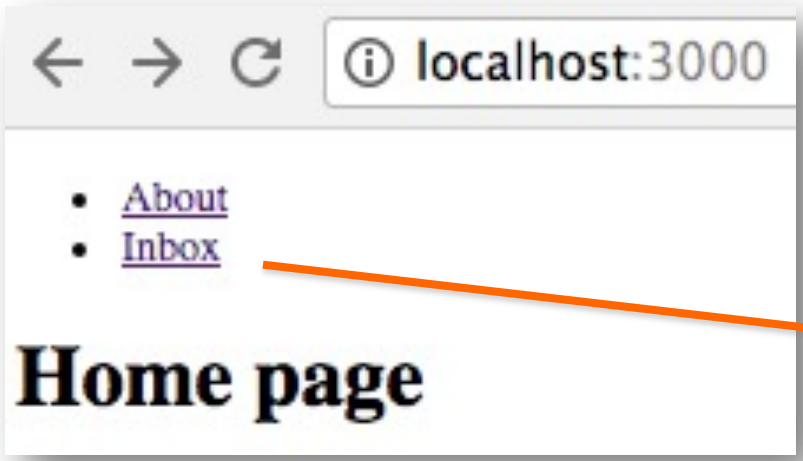
Nested Routes

```
<BrowserRouter>
  <Routes>
    <Route path="/about" element={<About />} />
    <Route path="/inbox/:userId" element={<Inbox />}>
      <Route path={`statistics`} element={<Stats />} />
      <Route path={`drafts`} element={<Draft />} />
      <Route index element={<Filler />} />
    </Route>
    <Route index element={<Home />} />
    <Route path="*" element={<Navigate to="/" replace />} />
  </Routes>
</BrowserRouter>
```

- **Use RELATIVE path strings for inner <Route> entries.**
- **The index <Route> is optional. It prevents ‘blank’ screen sections**

Extended <Link>

- **Objective: Pass additional props via a <Link>.**
- **EX.: See /src/sample5/.**



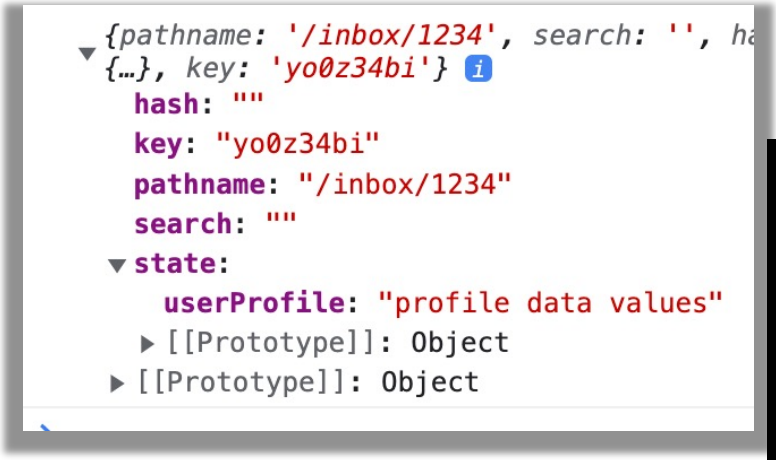

```
31    const userProfile = "profile data values";
32    return (
33      <>
34        <ul>
35          <li>
36            <Link to="/about">About</Link>
37          </li>
38          <li>
39            <Link
40              to={`/inbox/1234`}
41              state={{
42                userProfile:  userProfile,
43              }}
44            >
45              Inbox<span> (Link with extra props
46            </Link>
47          </li>
48        </ul>
```

```
<Route path="/inbox/:userId" element={<Inbox />}
```

- **How does Inbox access the userProfile?**

Extended <Link>

- **React Router creates a** location **object each time the URL changes.**
- **The** useLocation **hook enables access to Link data.**




```
14  const Inbox = (props) => {
15    const {userId} = useParams()
16    const locatio = useLocation();
17    console.log(locatio);
18    const {
19      state: { userProfile },
20    } = locatio;
21    return (
22      <>
23        <h2>Inbox page</h2>
24        <p>{`User Id: ${userId}`}</p>
25        <p>{`User profile: ${userProfile}`}</p>
26      </>
27    );
28  };
```

Routing

- **More later**