Agenda

- **Navigation** (Contd.)
- **Design patterns (Contd.)**
- **Global state.**
- **Complex State.**
- **Protected routes.**

Navigation

(Continued)

(See Archive from earlier lecture for code samples.)

Alternative <Route> API.

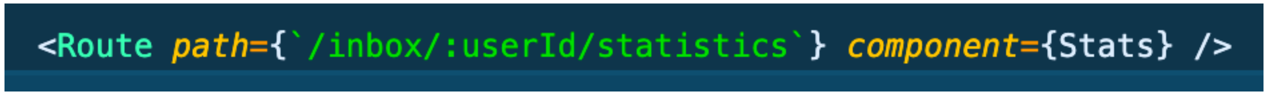
- **To-date:** *<Route path={…URL path…} component={ ComponentX} />*
  - **Mounted component always gets a default prop object.**
- **Disadv.: We cannot pass custom props to the mounted component.**

- **Alternative:**

  <Route path={…URL path…} render={…function….}>

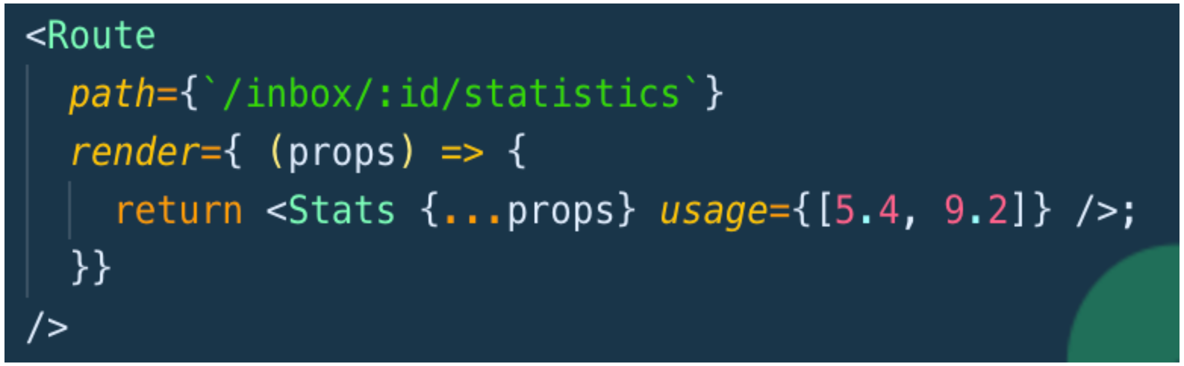
  – **where** *function* **return the mounted component.**
- **EX.: See** /src/sample7/**.**

  Objective: Pass usage data to the <Stats> **component from** sample4's nested Route.



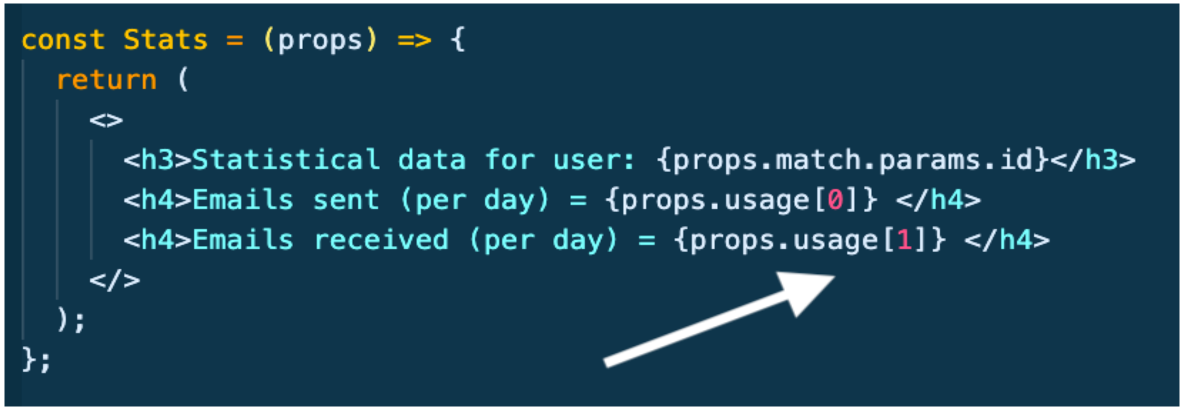
```
<Route path={`/inbox/:userId/statistics`} component={Stats} />
```

Alternative <Route> API.



```
<Route
  path={`/inbox/:id/statistics`}
  render={ (props) => {
    return <Stats {...props} usage={[5.4, 9.2]} />;
  }}
/>
```

- **The** render **prop function argument is the inherited props object.**

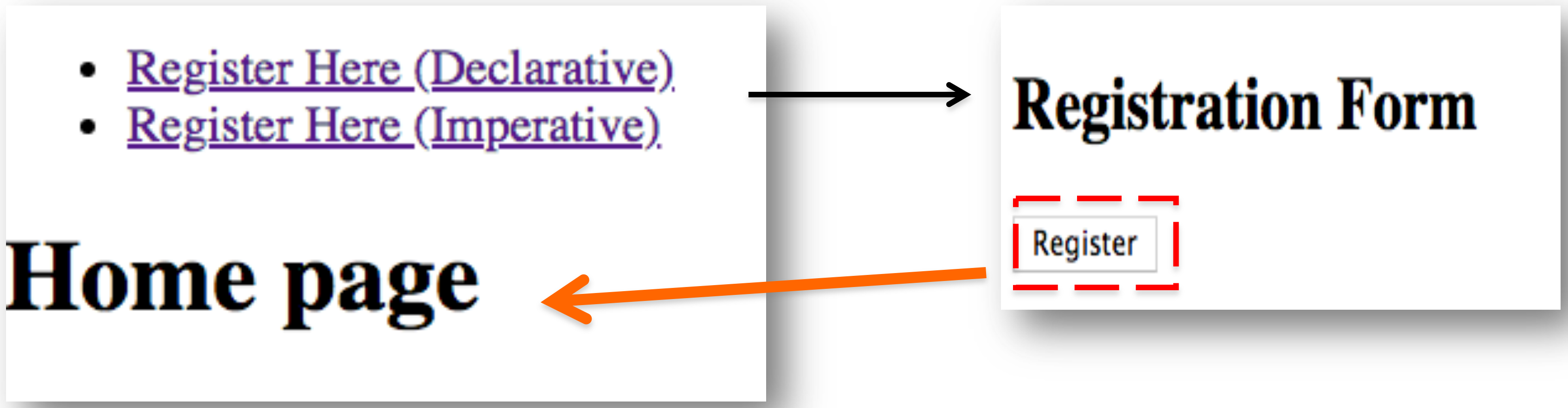


```
const Stats = (props) => {
  return (
    <>
      <h3>Statistical data for user: {props.match.params.id}</h3>
      <h4>Emails sent (per day) = {props.usage[0]} </h4>
      <h4>Emails received (per day) = {props.usage[1]} </h4>
    </>
  );
};
```

Programmatic Navigation.

- **Performing navigation in JavaScript.**
- **Two options:**
  1. **Declarative – requires state; use** <Redirect />.
  2. **Imperative – requires** withRouter**() ; use** props.history

- **EX.: See** /src/sample8/**.**



Summary

- **React Router package (version 4) adheres to React principles:**
  - **Declarative.**
  - **Component composition.**
  - **The event → state change → re-render**
- **Package's main components - <BrowserRouter>, <Route>, <Redirect>, <Link>.**
- **The** withRouter() **higher order component.**
- **Additional props:**
  - **props.match.params**
  - **props.history**
  - **props.location**

Design Patterns

(Continued)

Reusability & Separation of Concerns.

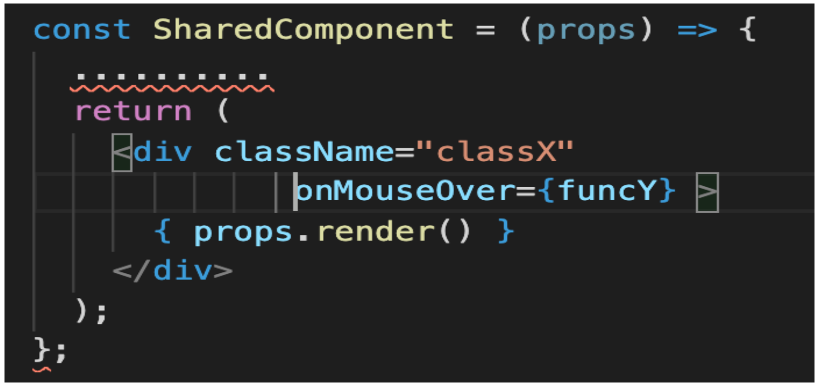
From previous lecture

- **Techniques to make cods reusable:**
  1. **Inheritance**
  2. **Composition**

- **React favors composition.**
- **Core React composition Patterns:**
  1. **Containers**
  2. **Render Props**
  3. **Higher Order Components.**

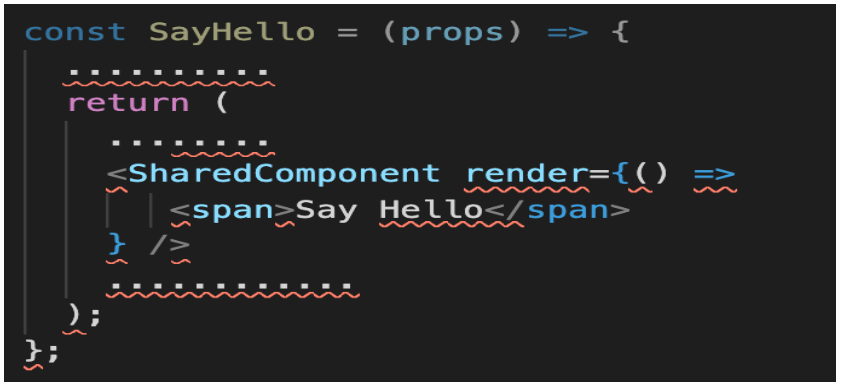
The Render Props pattern

- **Use the pattern to share logic between components.**
- **Dfn.**: A render prop is a function prop that a component uses to know what to render.

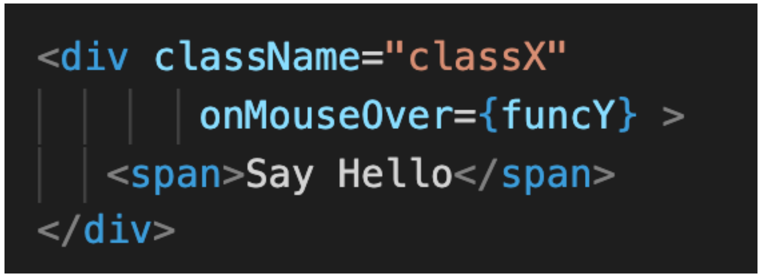


```
const SharedComponent = (props) => {
  ..........
  return (
    <div className="classX"
         onMouseOver={funcY} >
      { props.render() }
    </div>
  );
};
```

- SharedCoomponent **receives its render logic from the consumer, i.e.** SayHello.
- Prop name is arbitrary.



```
const SayHello = (props) => {
  ..........
  return (
    ........
    <SharedComponent render={() =>
      <span>Say Hello</span>
    } />
    .............
  );
};
```

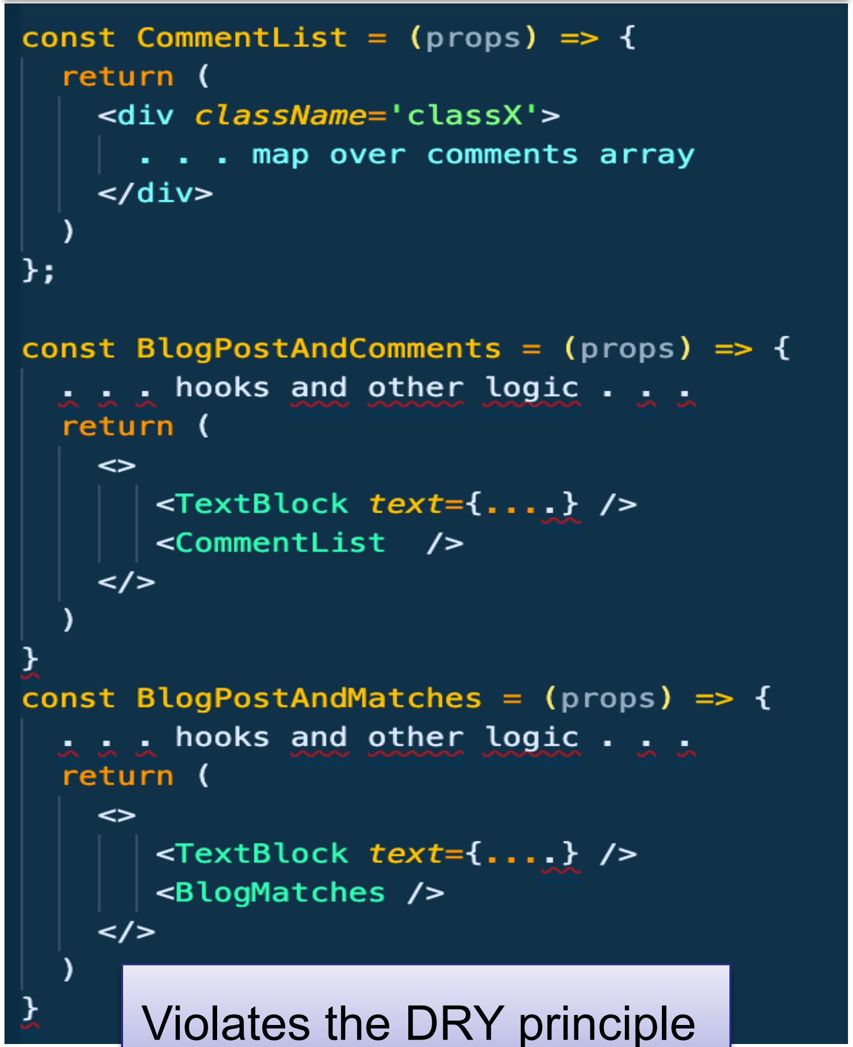


```
<div className="classX"
       onMouseOver={funcY} >
    <span>Say Hello</span>
</div>
```

The Render Props - Sample App.

- **A React app for viewing blog posts.**
  - **Suppose its views include:**
    1. **A view to display a post’s text followed by related comments.**
    2. **A view to display a post’s text followed by links to related / matching posts.**

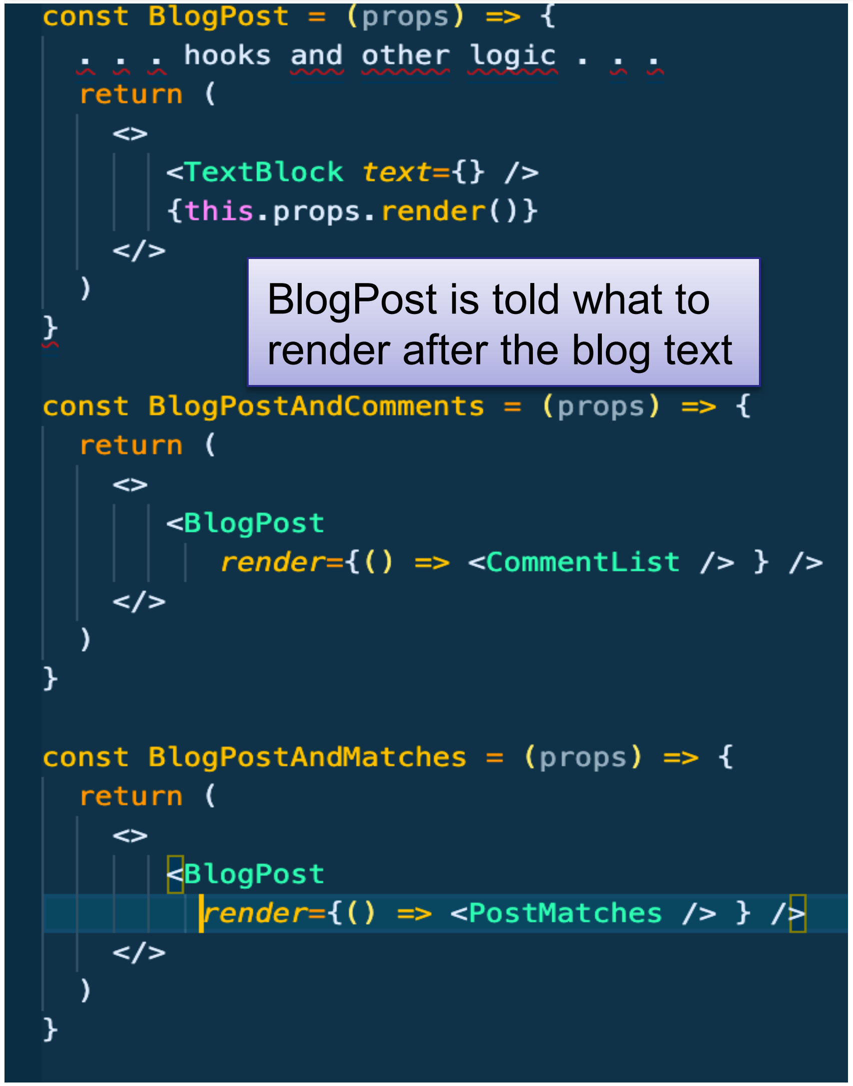
Without Render Props pattern



```
const CommentList = (props) => {
  return (
    <div className='classX'>
      . . . map over comments array
    </div>
  )
};

const BlogPostAndComments = (props) => {
  . . . hooks and other logic . . .
  return (
    <>
      <TextBlock text={....} />
      <CommentList  />
    </>
  )
}
const BlogPostAndMatches = (props) => {
  . . . hooks and other logic . . .
  return (
    <>
      <TextBlock text={....} />
      <BlogMatches />
    </>
  )
}
```

Violates the DRY principle

With Render Props pattern

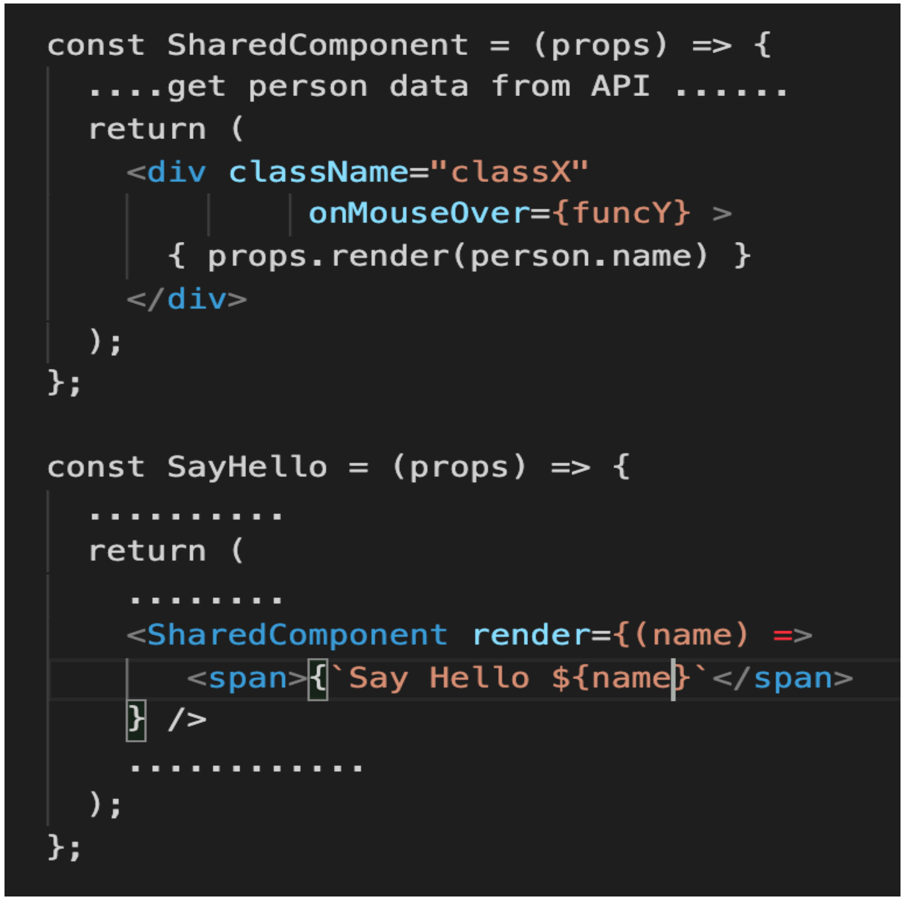


```
const BlogPost = (props) => {
  . . . hooks and other logic . . .
  return (
    <>
      <TextBlock text={} />
      {this.props.render()}
    </>
  )
}

const BlogPostAndComments = (props) => {
  return (
    <>
      <BlogPost
        render={() => <CommentList /> } />
    </>
  )
}

const BlogPostAndMatches = (props) => {
  return (
    <>
      <BlogPost
        render={() => <PostMatches /> } />
    </>
  )
}
```

BlogPost is told what to render after the blog text

The Render Props pattern

- **Render prop function can be parameterized**



```
const SharedComponent = (props) => {
  ....get person data from API ......
  return (
    <div className="classX"
            onMouseOver={funcY} >
      { props.render(person.name) }
    </div>
  );
};

const SayHello = (props) => {
  ..........
  return (
    ........
    <SharedComponent render={(name) =>
      <span>{`Say Hello ${name}`</span>
    } />
    ............
  );
};
```

- SharedCoomponent **generates the parameters required by the render prop function.**

Reusability.

- **Core React composition Patterns:**
  1. **Containers**
  2. **Render Props**
  3. **Higher Order Components.**

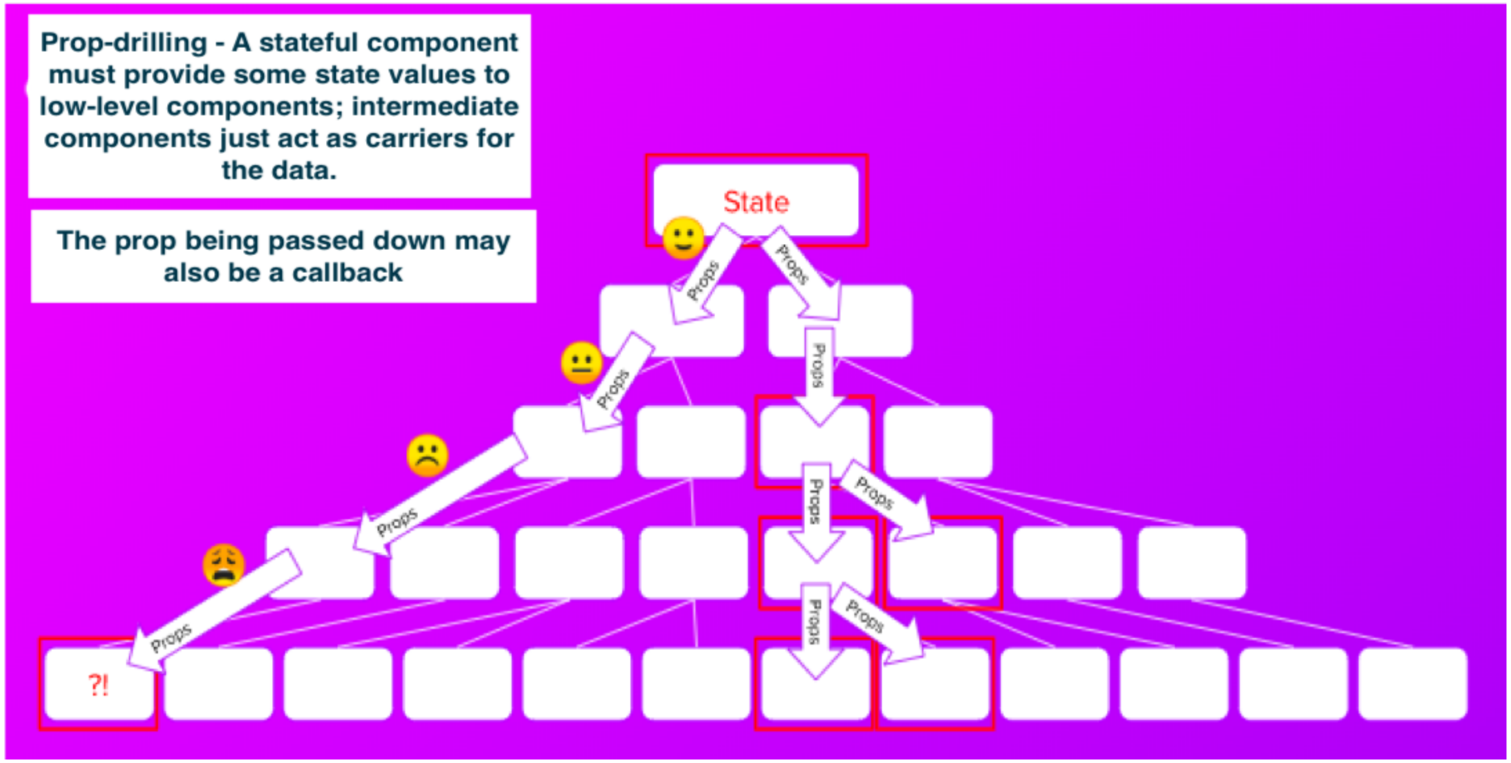
- **HOC is a function that takes a component and returns an enhanced version of it.**
  - **Enhancements could include:**
    - **Statefulness**
    - **Props**
    - **UI**
- **Ex –** withRouter **function.**

React Contexts

The Provider pattern

The Provider pattern – When?

- **Use cases:**
  1. **Sharing** global data/state**, e.g. Web API data.**
  2. **To avoid** prop-drilling**.**

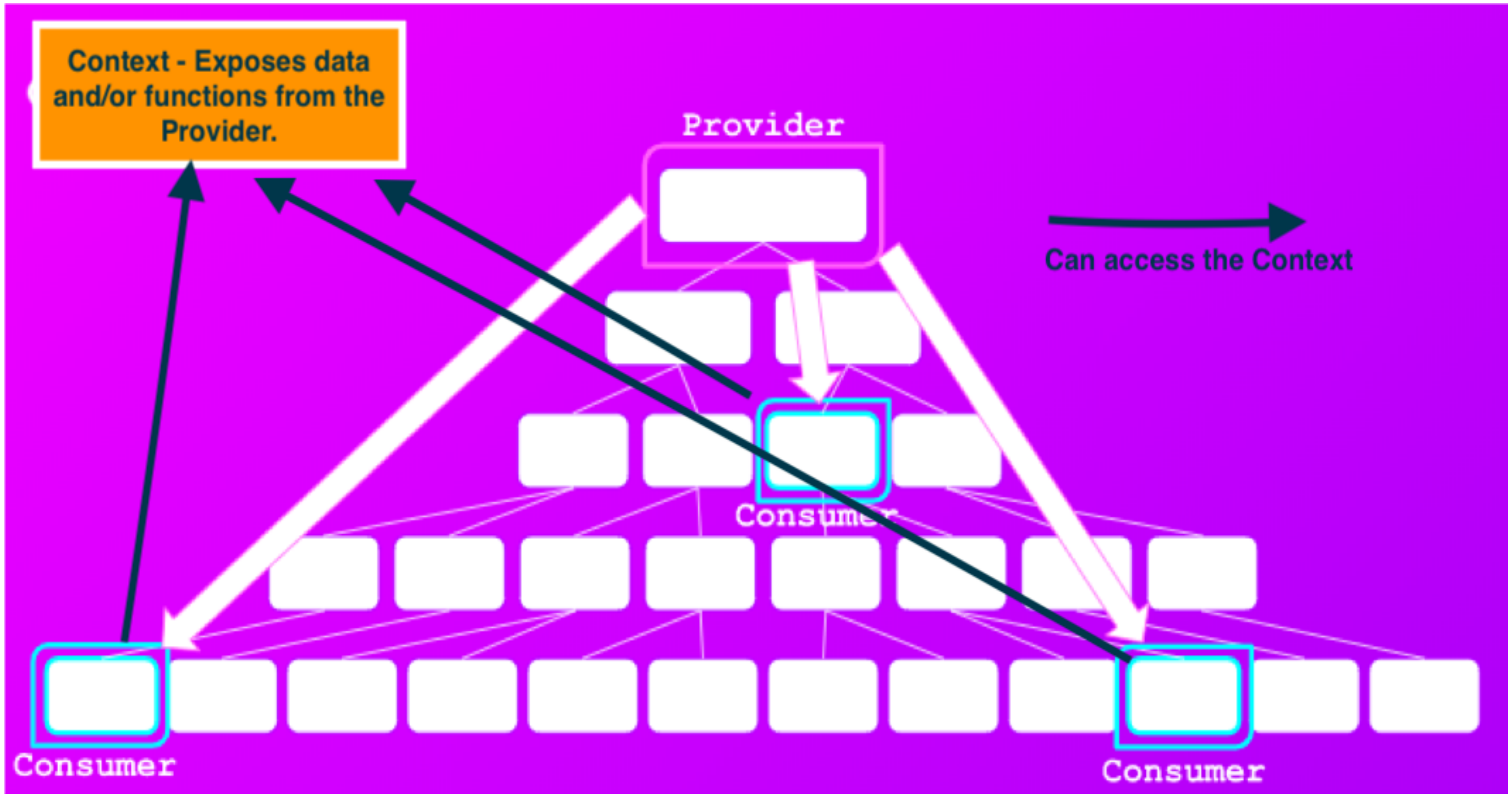

Prop-drilling - A stateful component must provide some state values to low-level components; intermediate components just act as carriers for the data.
The prop being passed down may also be a callback
State
Props
?!

The Provider pattern – How?

- **React Implementation steps:**
  1. **Declare a component for managing the global data – the Provider component.**
  2. **Make the data accessible by other components (consumers) – using** Contexts.
  3. **Use component composition to integrate the Provider with consumer(s).**

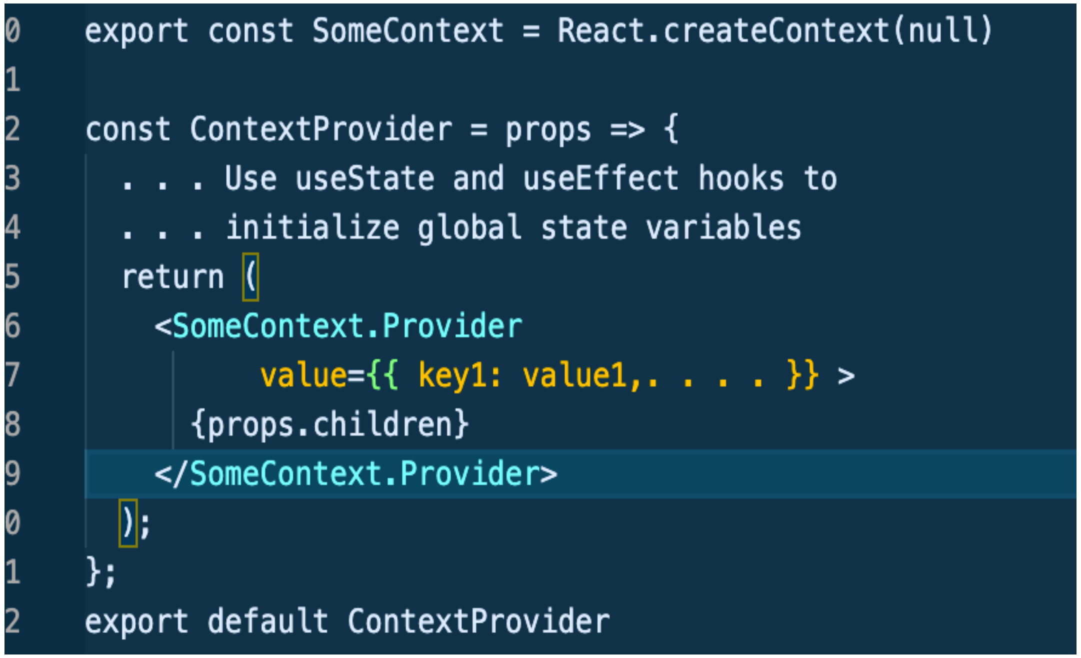
- Contexts **– the glue behind provider pattern in React.**
  – **A** Context **provides a way to pass data through the component hierarchy without having to pass props down manually at every level.**
  – **Provider component creates/manages the context.**
  – **Consumer accesses context with** useContext **hook**

The Provider pattern – React Contexts.



The Provider pattern – Implementation

- **Declare the Provider component:**

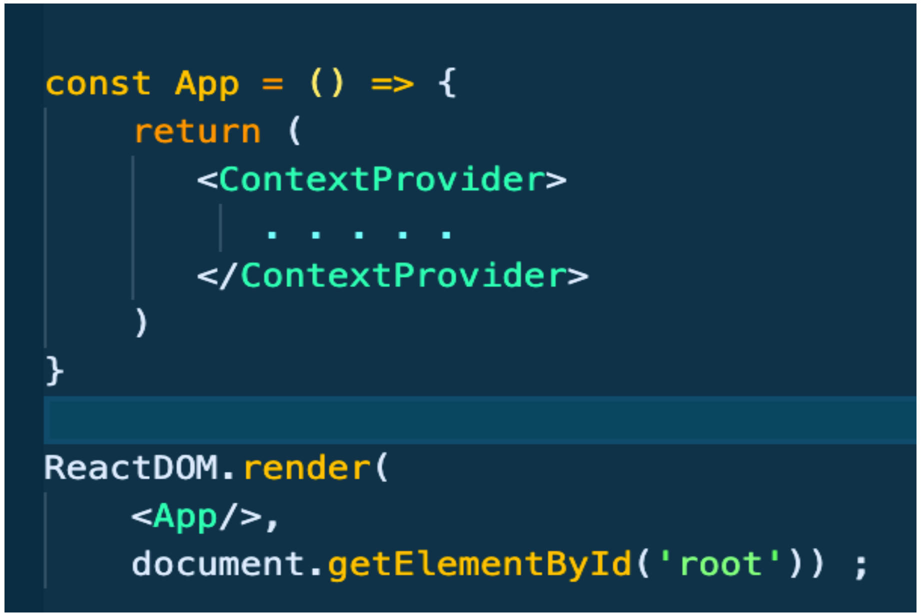


```
export const SomeContext = React.createContext(null)

const ContextProvider = props => {
  . . . Use useState and useEffect hooks to
  . . . initialize global state variables
  return (
    <SomeContext.Provider
          value={{ key1: value1,. . . . }} >
      {props.children}
    </SomeContext.Provider>
  );
};
export default ContextProvider
```

- **We associate the** Context **with the** Provider **component using** <contextName.Provider>.
- **The** values object **declares what is accessible by consumers.**
  - **Functions as well as state data can be values.**

The Provider pattern – Implementation.

- **Integrate (Compose) the Provider with the rest of the app using the Container pattern**

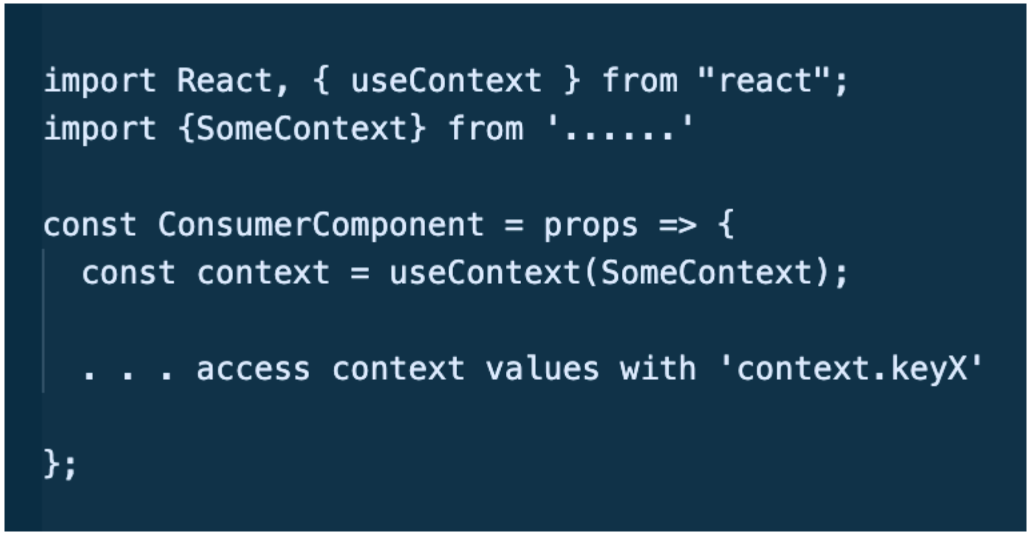


```
const App = () => {
    return (
       <ContextProvider>
          . . . . .
       </ContextProvider>
    )
}

ReactDOM.render(
    <App/>,
    document.getElementById('root')) ;
```

- **The Provider's** children **will now be able to access the shared context.**

The Provider pattern – Implementation.

- **The context users accesses the context with** useContext **hook.**



```
import React, { useContext } from "react";
import {SomeContext} from '......'

const ConsumerComponent = props => {
  const context = useContext(SomeContext);

  . . . access context values with 'context.keyX'

};
```

- **The** context's key**s match those of the** values **object exposed by the Provider component.**

The Provider pattern.

- **When not to use Contexts:**
  1. **To avoid ‘shallow’ prop drilling. Prop drilling is faster for this case.**
  2. **To save state that should be kept locally, e.g. web form inputs should be in local state.**
  3. **For large object as context value - monitor performance and refactor as necessary.**

useReducer hook

(See archive for full source code)

useReducer hook

- **“useReducer** is preferable to **useState** when you have complex state logic that involves non-primitive state values or when the next state is computed from the previous one.”

- **Based on the Redux 3rd party library for managing application state.**
  - **The M part of MVC.**

- **Signature:**

  const [state, ***dispatch***] = useReducer(***reducer***, initial state)

- **Constructs** – State (non-primitive), Reducer function, Actions, Dispatcher.

useReducer elements.

- reducer **– A custom function for mutating the state:**

  reducerFunction(currentState, action) → newState

  – **currentState is immutable.**
  – **Must make a copy of currentState and change the copy.**
  – **Must return the (mutated) copy.**
- **Reducer** actions **– An object:** { actionType, payload }
  – **Reducer function uses an action's** type **to determine the nature of the state change required. The** payload **is the data applied in a state change.**
- **dispatch - function to send/dispatch actions to the reducer.**

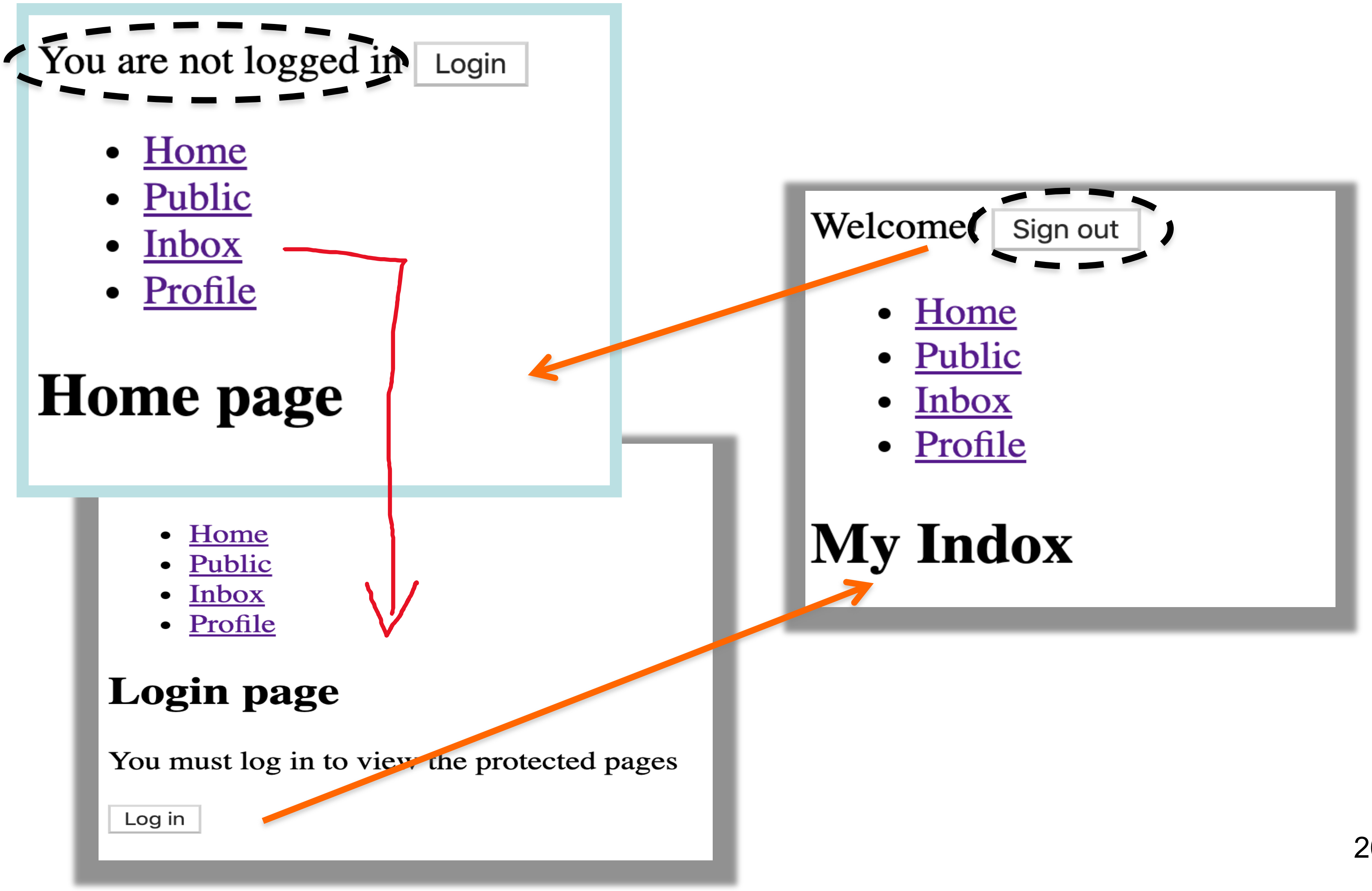
  dispatcher(action)

- **The Zoom party manager - See archive sample app**

Authentication and Protected/Private Routes

(See Routing samples Archive)

Objective



You are not logged in Login

- Home
- Public
- Inbox
- Profile

Home page

- Home
- Public
- Inbox
- Profile

Login page

You must log in to view the protected pages

Log in

Welcome! Sign out

- Home
- Public
- Inbox
- Profile

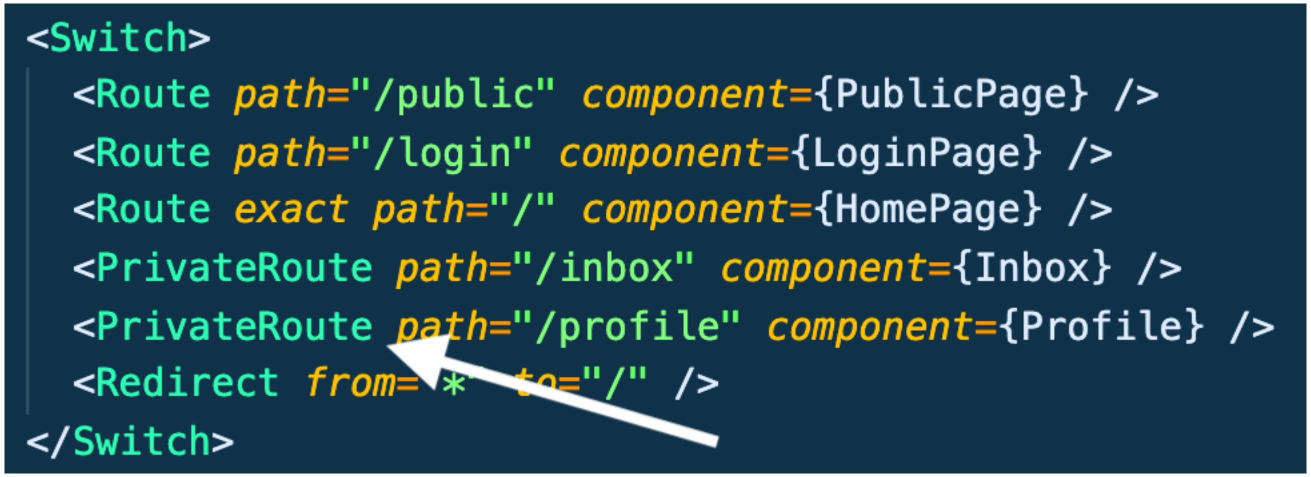
My Indox

Protected Routes.

- **Not native to React Router.**

- **We need a custom solution.**

- **Solution outline: Clear, declarative style for declare views/pages requiring authentication:**



```
<Switch>
  <Route path="/public" component={PublicPage} />
  <Route path="/login" component={LoginPage} />
  <Route exact path="/" component={HomePage} />
  <PrivateRoute path="/inbox" component={Inbox} />
  <PrivateRoute path="/profile" component={Profile} />
  <Redirect from="*" to="/" />
</Switch>
```

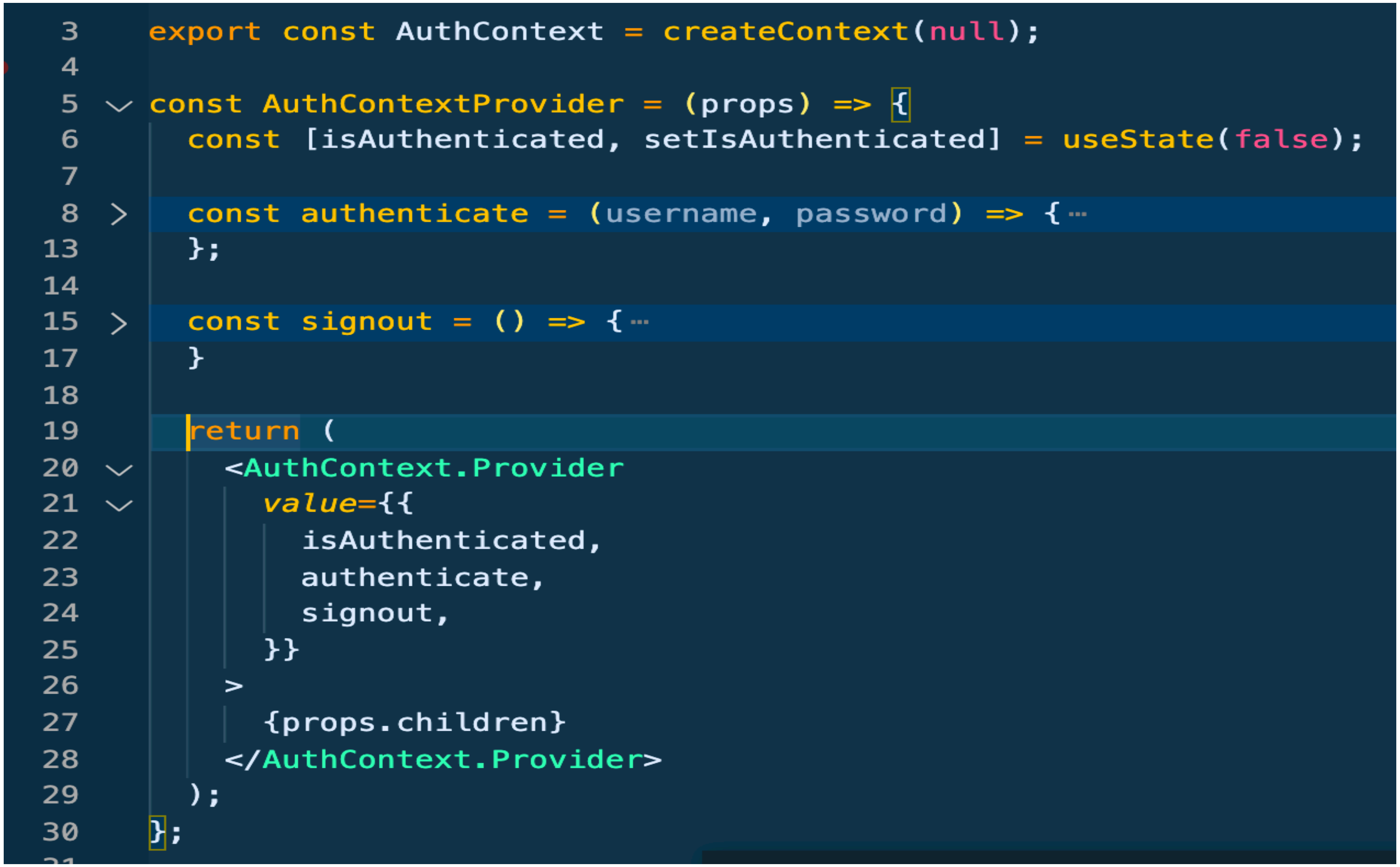
Protected Routes.

- **Solution features:**

  1. **React Context to store current authenticated user.**
  2. **Programmatic navigation - to redirect unauthenticated user to login page.**
  3. **Remember user’s intent before forced authentication.**

Protected Routes

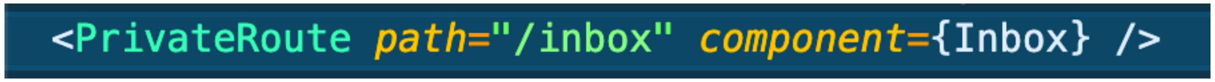
- **Solution elements: The AuthContext.**



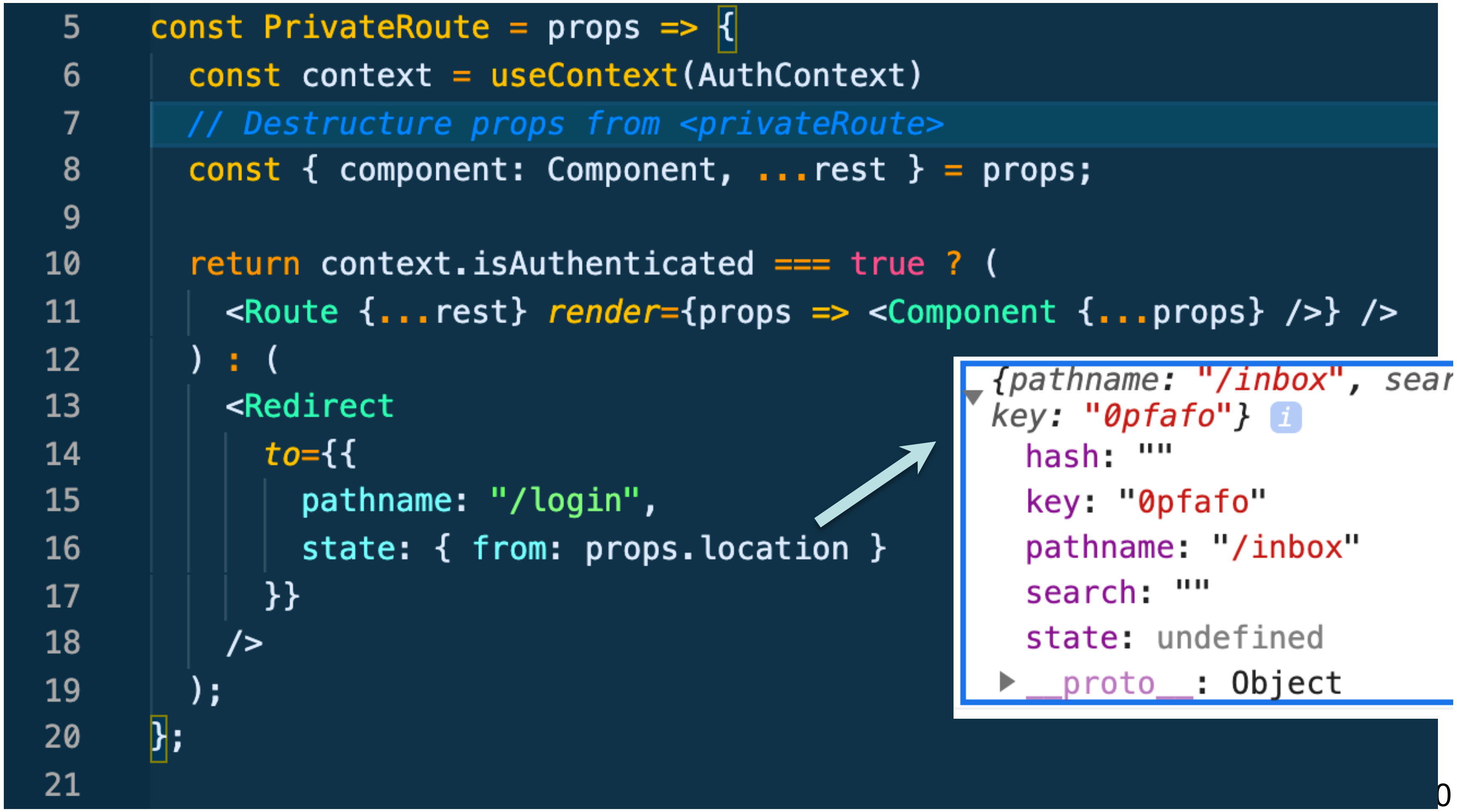
```
export const AuthContext = createContext(null);

const AuthContextProvider = (props) => {
  const [isAuthenticated, setIsAuthenticated] = useState(false);

  const authenticate = (username, password) => {⋯
  };

  const signout = () => {⋯
  }

  return (
    <AuthContext.Provider
      value={{
        isAuthenticated,
        authenticate,
        signout,
      }}
    >
      {props.children}
    </AuthContext.Provider>
  );
};
```

Protected Routes

- **Solution elements (Contd.): <PrivateRoute />**



```
<PrivateRoute path="/inbox" component={Inbox} />
```

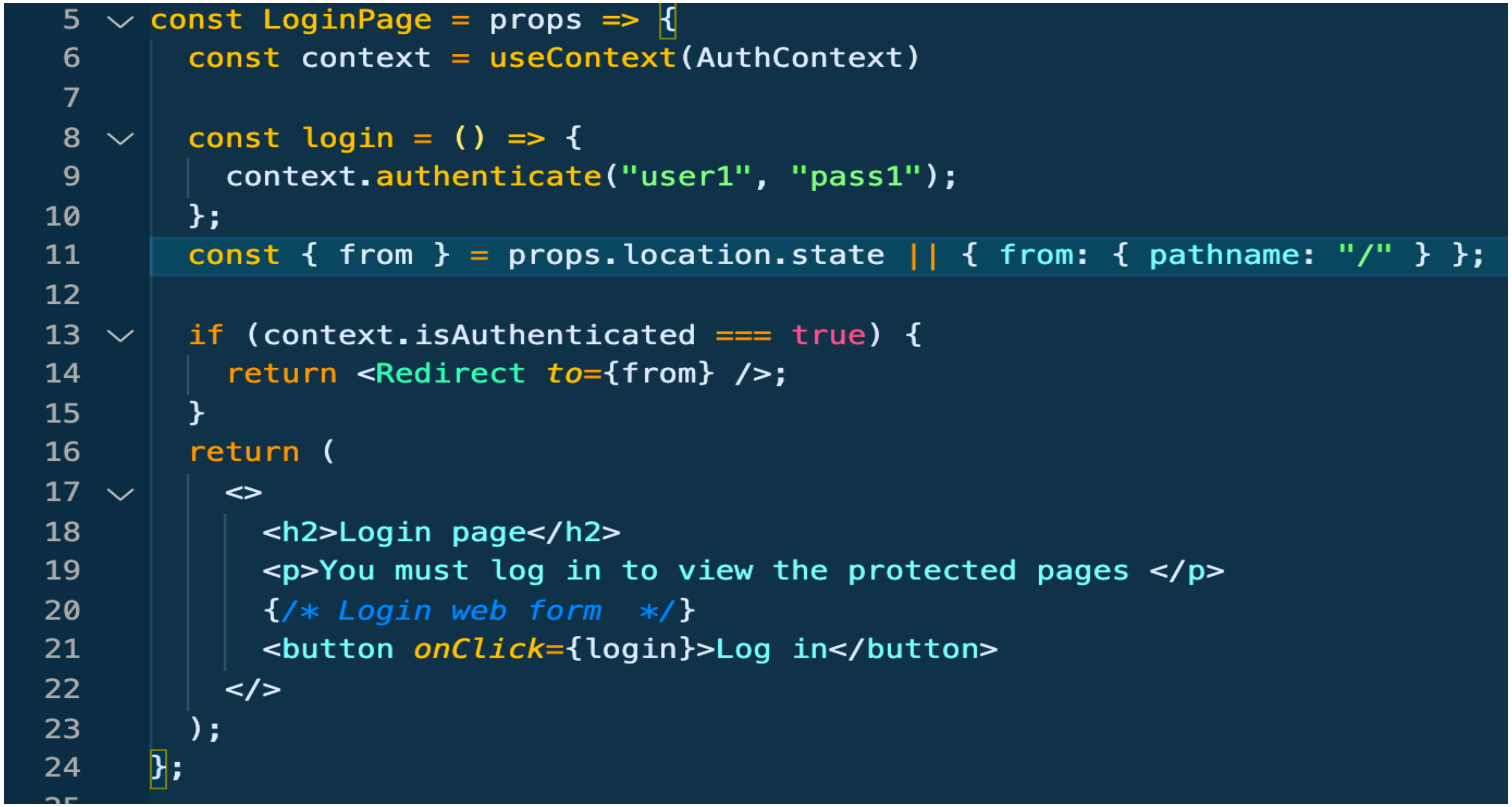


```
const PrivateRoute = props => {
  const context = useContext(AuthContext)
  // Destructure props from <privateRoute>
  const { component: Component, ...rest } = props;

  return context.isAuthenticated === true ? (
    <Route {...rest} render={props => <Component {...props} />} />
  ) : (
    <Redirect
      to={{
        pathname: "/login",
        state: { from: props.location }
      }}
    />
  );
};
```

```
▾ {pathname: "/inbox", sear
  key: "0pfafo"}
    hash: ""
    key: "0pfafo"
    pathname: "/inbox"
    search: ""
    state: undefined
  ▸ __proto__: Object
```

Protected Routes

- **Solution elements (Contd.): <LoginPage>**



```
const LoginPage = props => {
  const context = useContext(AuthContext)

  const login = () => {
    context.authenticate("user1", "pass1");
  };
  const { from } = props.location.state || { from: { pathname: "/" } };

  if (context.isAuthenticated === true) {
    return <Redirect to={from} />;
  }
  return (
    <>
      <h2>Login page</h2>
      <p>You must log in to view the protected pages </p>
      {/* Login web form  */}
      <button onClick={login}>Log in</button>
    </>
  );
};
```