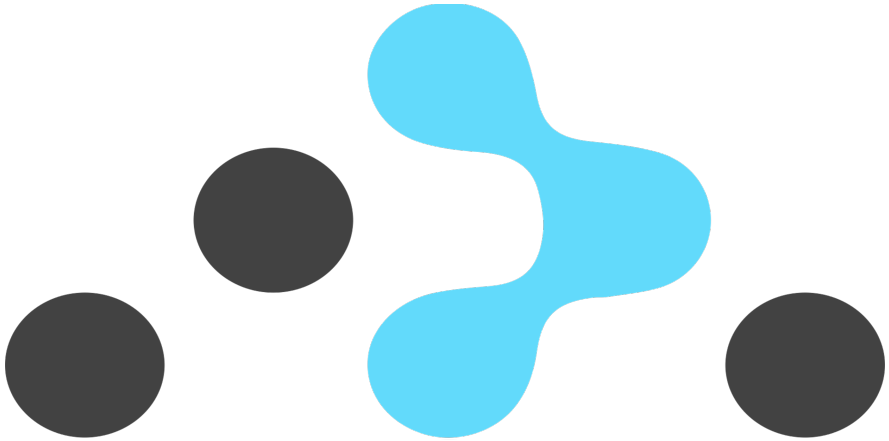
Agenda.

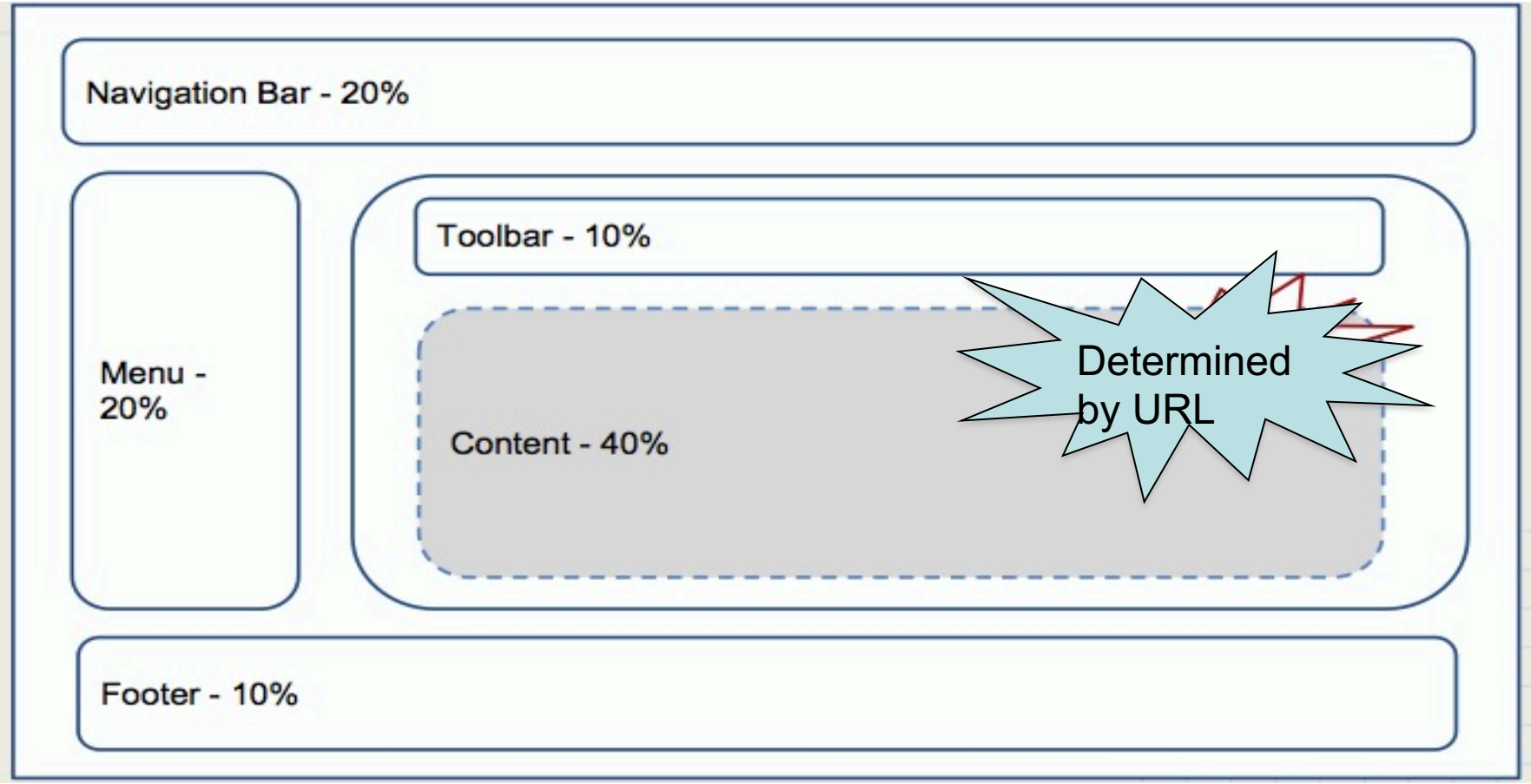
- **Navigation / Routing (Contd.)**
- **Design Patterns – The Provider Pattern**
- **Protected/Private Routes.**
  - **A use case for the Provider pattern and many of the routing techniques.**



Navigation

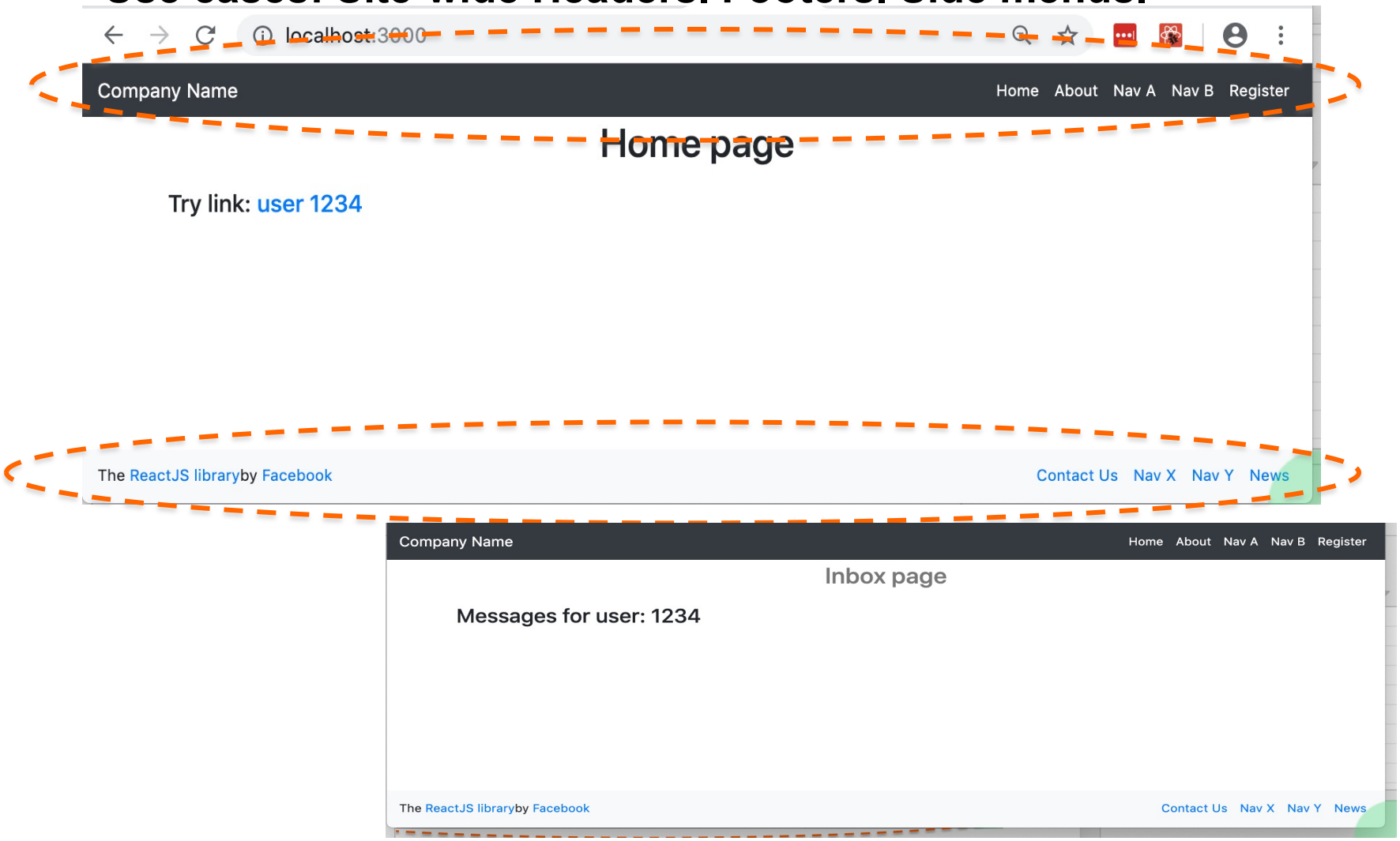
(Continued)

Typical Web app layout



Persistent elements/components

- **Use cases: Site-wide Headers. Footers. Side menus.**



Persistent elements/components

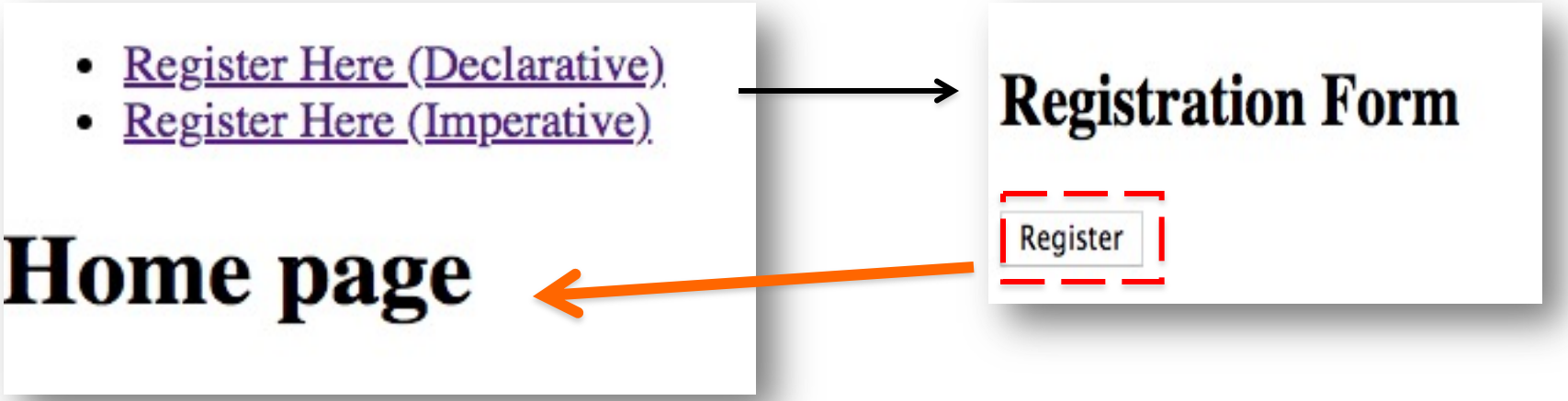
- **Ref.** src/sample6

```
27  const App = () => {
28    return (
29      <BrowserRouter>
30        <Header />
31        <div className="container">
32          <Routes>
33            <Route path="/about" element={<About />} />
34            <Route path="/register" element={<Register />} />
35            <Route path="/contact" element={<Contact />} />
36            <Route path="/inbox/:userId" element={<Inbox />} />
37            <Route index element={<Home />} />
38            <Route path="*" element={<Navigate to="/" replace />} />
39          </Routes>
40        </div>
41        <Footer />
42      </BrowserRouter>
43    );
44  };
```

Programmatic Navigation.

- **Perform navigation a component's JS logic.**
- **Two options:**
  1. **Declarative –custom state variable and** <Navigate />.
  2. **Imperative –** the useNavigate **hook**

- **EX.: See** /src/sample7/**.**



Routing Summary

- **React Router package adheres to React principles:**
  - **Declarative.**
  - **Component composition.**
  - **The event → state change → re-render**
- **Package's main components - <BrowserRouter>, <Route>, <Navigate>, <Link>.**

- **Special hooks to allow us access routing data/methods, e.g.** useParams, useNavigate, useLocation.

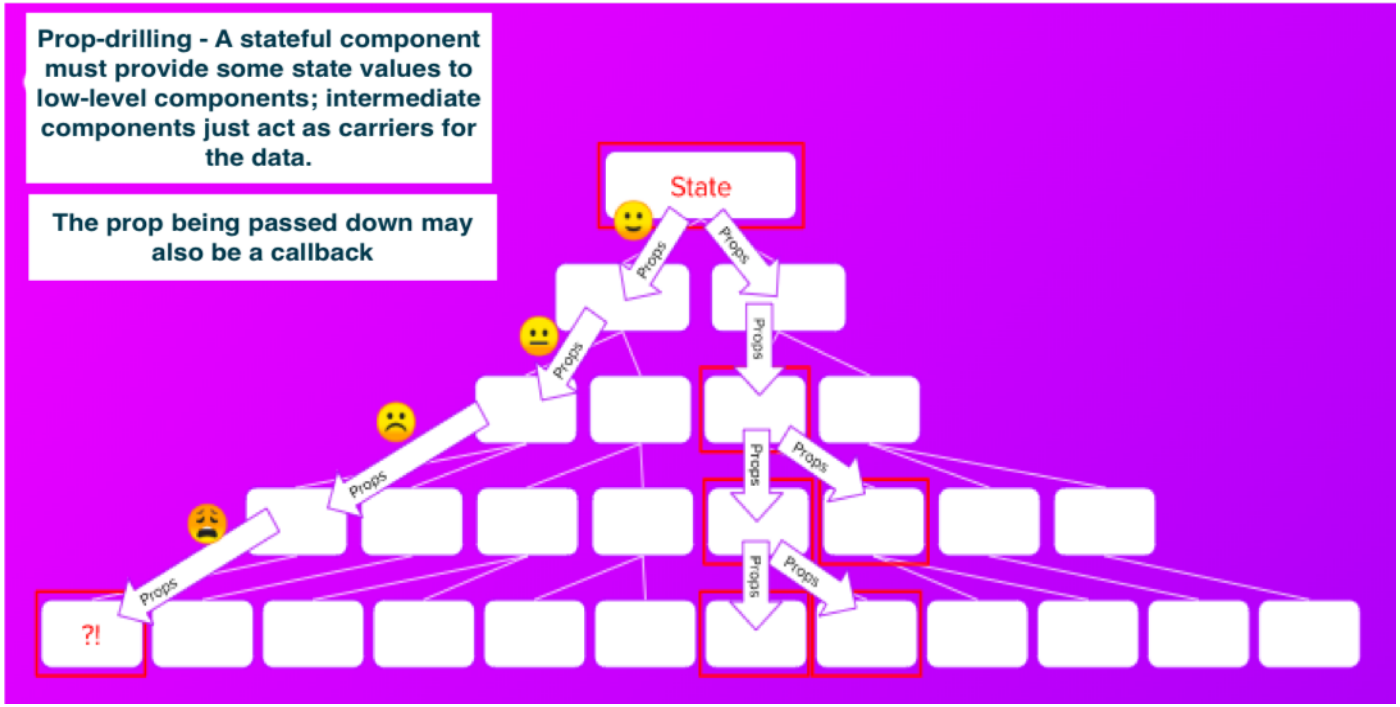
Design Patterns

(Contd.)**.**

The Provider pattern - React Context

The Provider pattern – When?

- **Use cases:**
  1. Sharing data/state **with multiple components**, i.e. global data**, e.g. favourite movies.**
  2. **To avoid** prop-drilling**.**

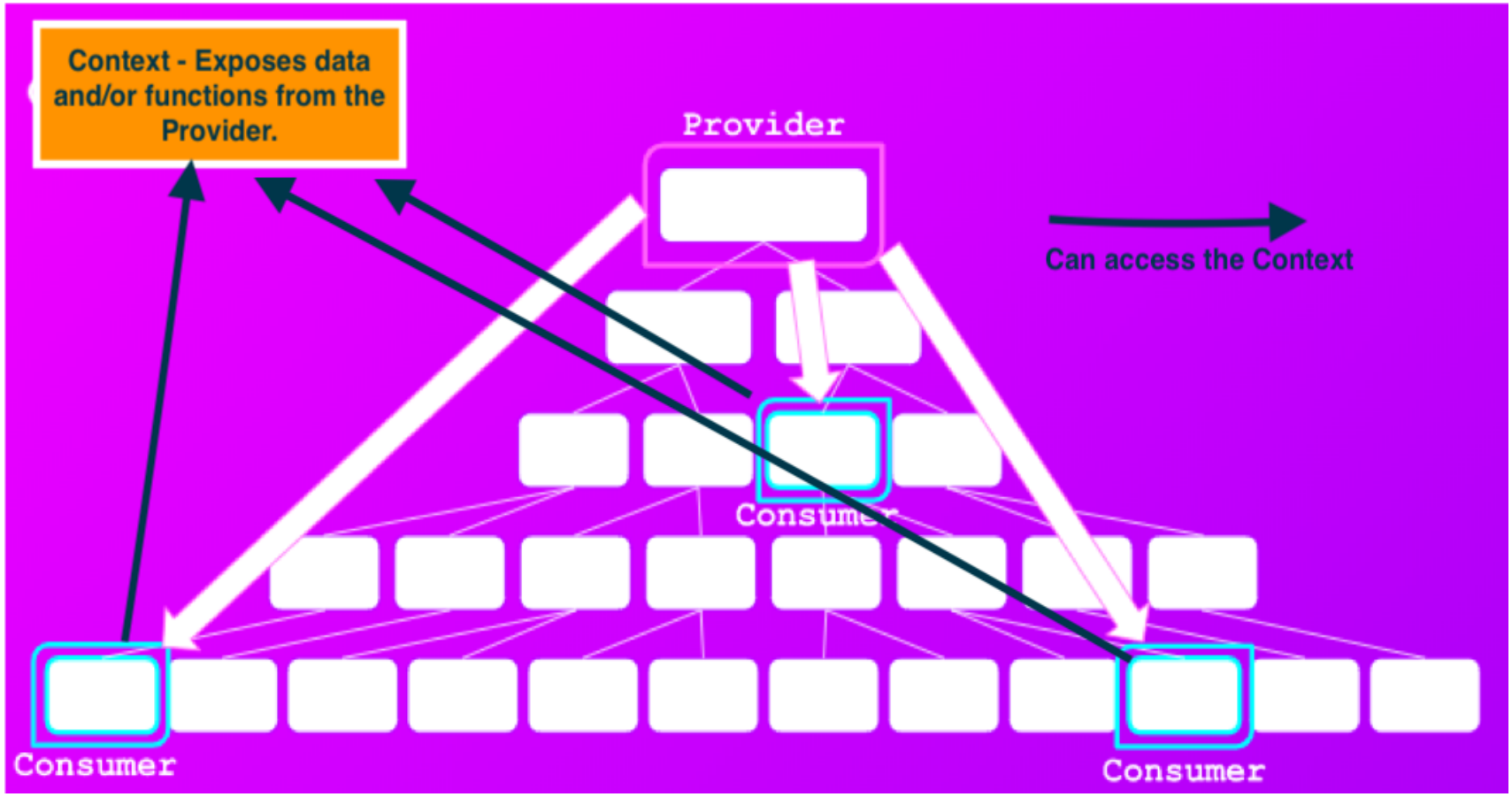


The Provider pattern – How?

- **React Implementation steps:**
  1. **Declare a component to manage the shared data – the** Provider **component.**
  2. **Create a** context **construct and link it to the Provider.**
  3. **Wrap the shared data in the context.**
  4. Compose **the Provider with other components to allow them access the context.**

- Context **– the glue for the Provider pattern in React.**
  - **Avoids** prop drilling**.**
  - **Provider component manages the context.**
  - **Consumer's access the context with the** useContext **hook**

The Provider pattern – React Context.



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 link the** Context **to the** Provider component **using** <contextName.Provider>.
- **The** values object **declares the context's content. .**
  - **Can be functions (behaviour) as well as data (state).**

The Provider pattern – Implementation.

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

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

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

- **All the app's pages can now access the context.**

The Provider pattern – Implementation.

- useContext **hook – givss a component access to a contex:/**

  const contextRef = useContext(ContextName)

  // contextRef points at context's values object.

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

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

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

};
```

```
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
```

The Provider pattern – Implementation.

- **For improved** separation of concerns**, use multiple context instead of a 'catch all' context.**

```
const App = () => {
    return (
        <BrowserRouter>
          <ContextProviderA>
            <ContextProviderB>
              . . . . . .
            </ContextProviderB>
          </ContextProviderA>
        </BrowserRouter>
    );
  };
```

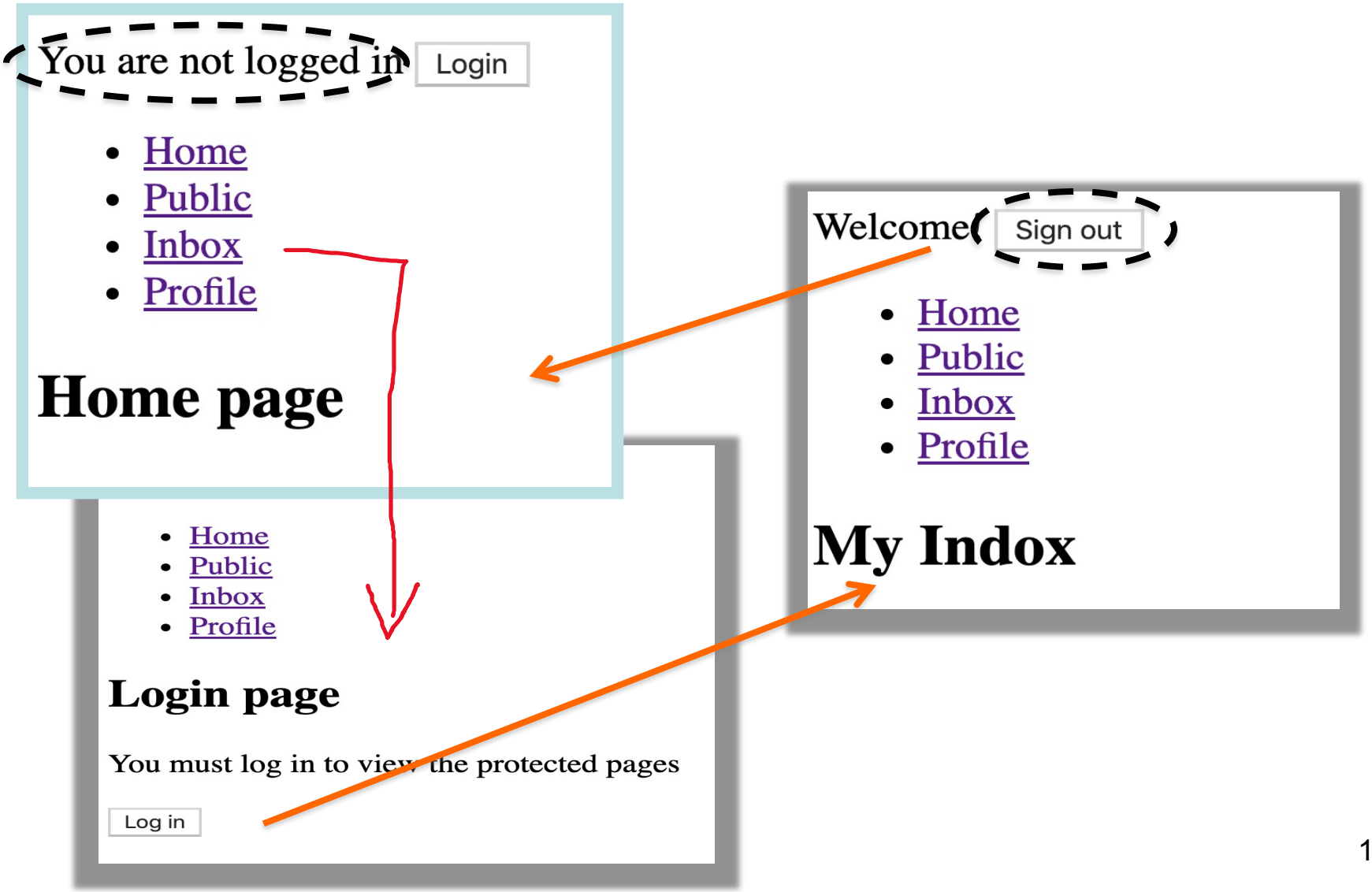
The Provider pattern.

- **When NOT to use a Context:**
  1. **To avoid ‘shallow’ prop drilling.**
     - **Prop drilling is faster for ‘shallow’ cases.**
  2. **For state that should be kept local to a component, e.g. web form inputs.**
  3. **For large object - monitor performance and refactor as necessary.**

Authentication and Protected/Private Routes

(See Routing samples Archive)

Objective


You are not logged in
Login
Home
Public
Inbox
Profile
Home page
Welcome
Sign out
Home
Public
Inbox
Profile
My Indox
Home
Public
Inbox
Profile
Login page
You must log in to view the protected pages
Log in

Protected Routes - Solution outline.

- **Not native to React Router.**
- **We need a custom solution.**
- Solution outline**: A clear, declarative style for declare the views/pages that require authentication.**

```
<Routes>
  <Route path="/public" element={<PublicPage />} />
  <Route path="/login" element={<LoginPage />} />
  <Route index element={<HomePage />} />
  <Route path="/inbox"element={
      <ProtectedRoute>
        <Inbox />
      </ProtectedRoute>
    }
  />
  <Route path="/profile" element={
      <ProtectedRoute>
        <Profile />
      </ProtectedRoute>
    }
  />
  <Route path="*" element={<Navigate to="/" replace />} />
</Routes>
```

Solution elements.

- **Solution features:**

  1. **A React Context to store the current authenticated user's token.**
  2. **Programmatic navigation - to redirect unauthenticated user to login page.**
  3. **Remember user's intent prior to the forced authentication step.**

Implementation

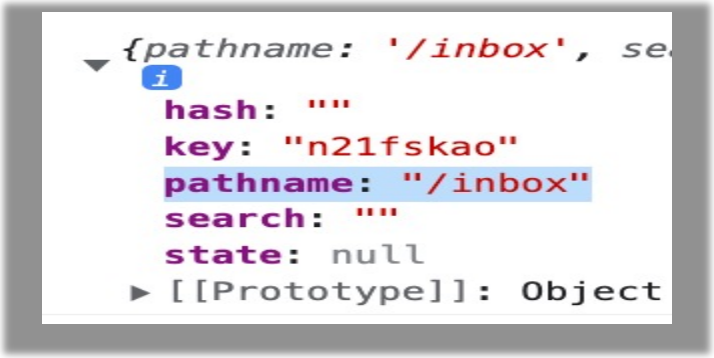
- **Solution elements: The AuthContext.**

```
 4    export const AuthContext = createContext(null);
 5
 6    const AuthContextProvider = ({ children }) => {
 7      const [token, setToken] = useState(null);
 8      const location = useLocation();
 9      const navigate = useNavigate();
10
11      const authenticate = (username, password) => {
12        setTimeout(() => {
13          setToken("2342f2f1d131rf12");
14          const origin = location.state?.intent || "/";
15          navigate(origin);
16        }, 250);
17      };
18
19  >   const signout = () => {…
22      };
23
24      return (
25        <AuthContext.Provider
26          value={{ token, authenticate, signout }}
27        >
28          {children}
29        </AuthContext.Provider>
30      );
31    };
```

Implementation

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

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




```
5   const ProtectedRoute = ({ children }) => {
6     const { token } = useContext(AuthContext);
7     const location = useLocation();
8     // console.log(location)
9     if (!token) {
10      return <Navigate to={"/login"} replace
11                       state={{ intent: location.pathname }} />;
12    }
13
14    return children;
15  };
16
```

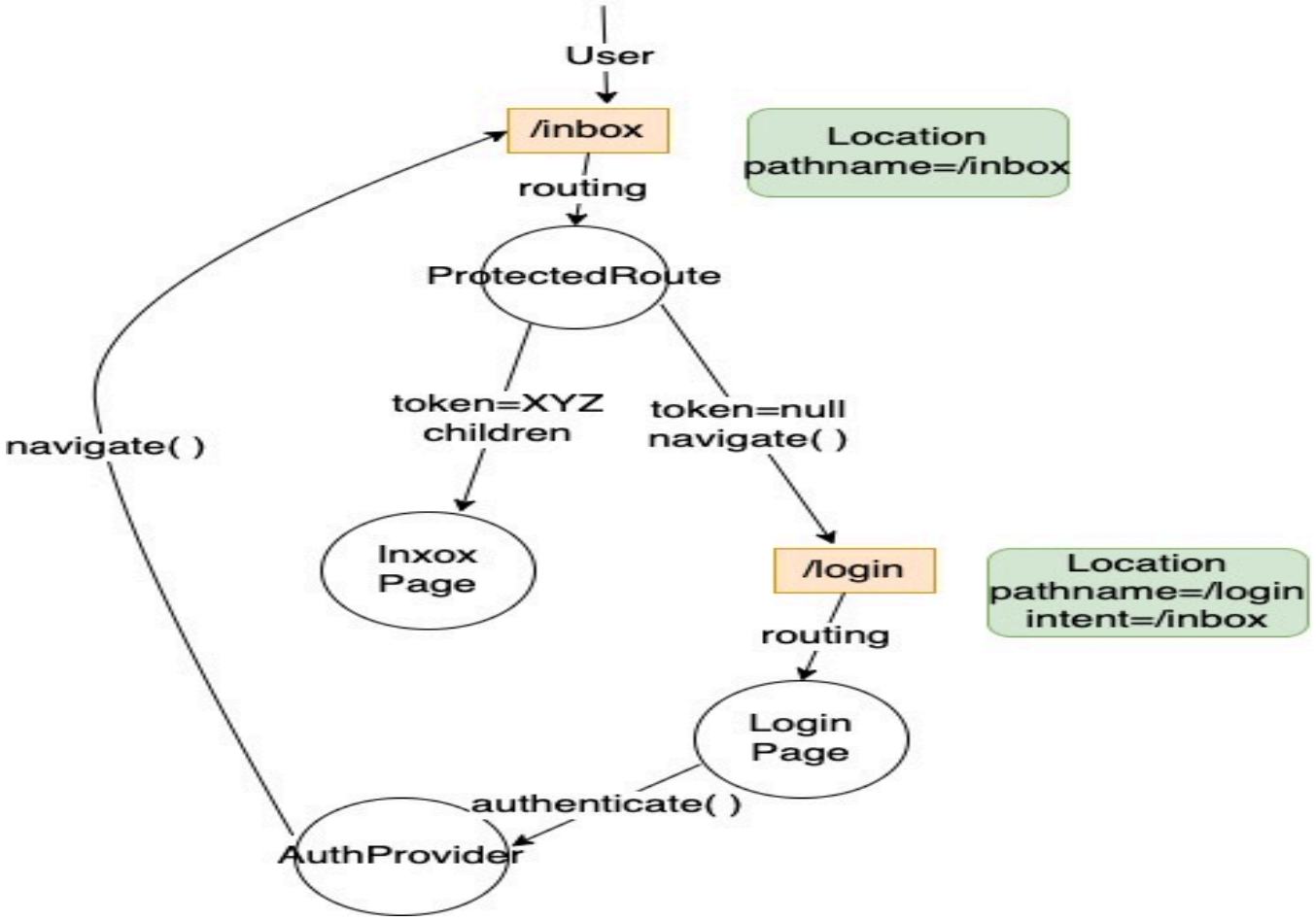
Implementation

- **Solution elements (Contd.): The Login Page.**

```
4   const LoginPage = () => {
5     const {authenticate } = useContext(AuthContext);
6
7     const login = () => {
8       const password = Math.random().toString(36).substring(7);
9       authenticate('user1', password);
10    };      You, 3 weeks ago • Upgrade to React Router v6 - Major
11
12    return (
13      <>
14        <h2>Login page</h2>
15        <p>You must log in to view the protected pages </p>
16        {/* Login web form  */}
17        <button onClick={login}>Submit</button>
18      </>
19      )
20  };
```

Implementation - Flow of control.

When an unauthenticated user tries to access /inbox



The optional chaining operator (?.)

- **The optional chaining operator (?.) accesses an object's property. If the property is undefined or null, the expression short circuts and evaluates to undefined instead.**

```
1   let var1 = {}  // Empty object
2   let var2 = var1.foo // undefined
3   let var3 = var1.foo.bar // Runtime ERROR
4   let var4 = var1.foo?.bar // undefined
5   let var5 = var1.foo?.bar?.baz // undefined
6
7   var1 = {foo: {bar: 10}}
8   var4 = var1.foo?.bar // 10
9
```

The code archive.

- **Two implementations:**

1. **Version 1 - AuthContext and login page only; No ProtectedRoute or Remember intent.**
2. **Version 2 – Full implementation.**

- **The fakeAuth() function and the async/await model for asynchronous programming.**