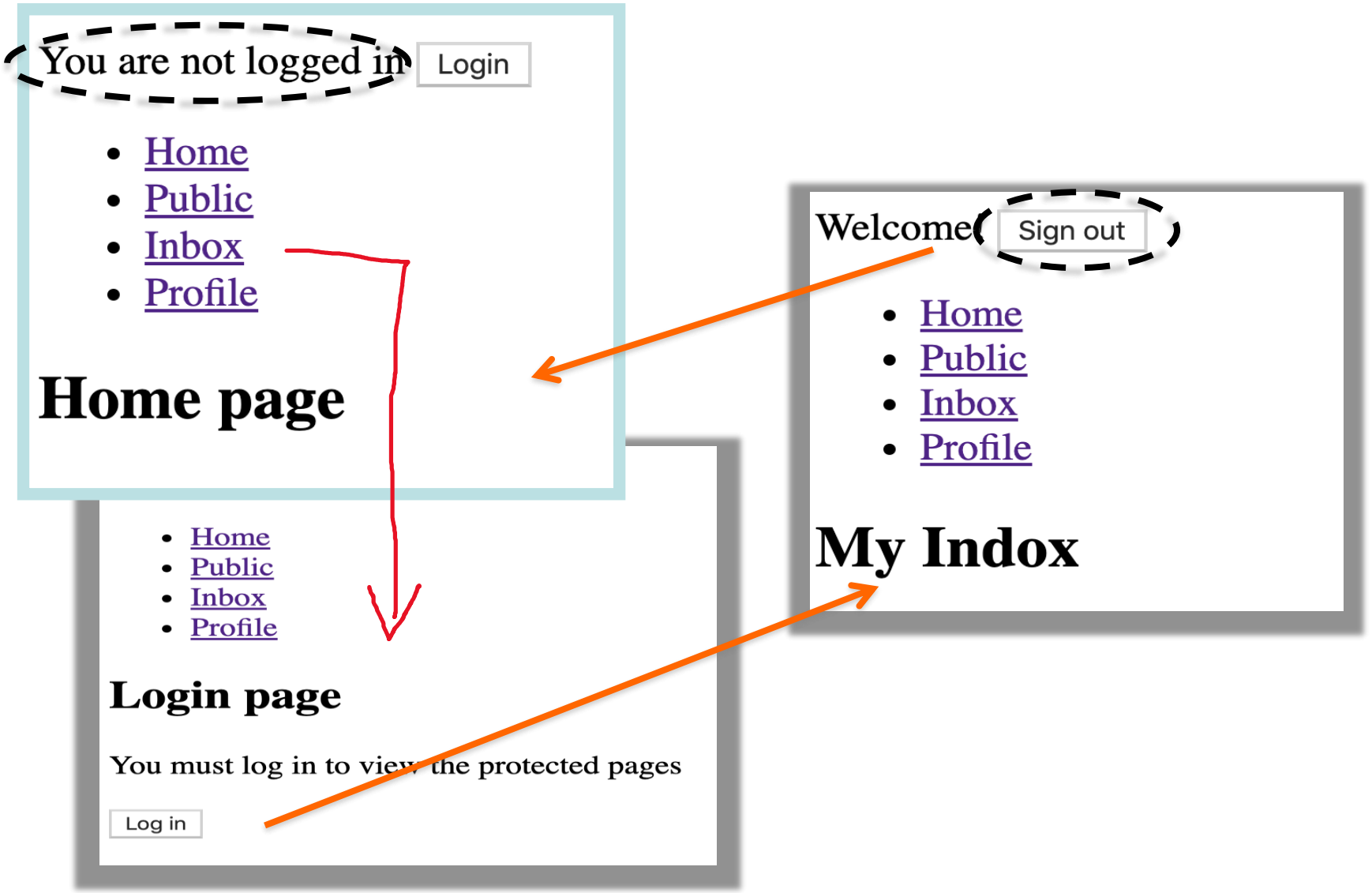
Authentication and Protected/Private Routes

(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

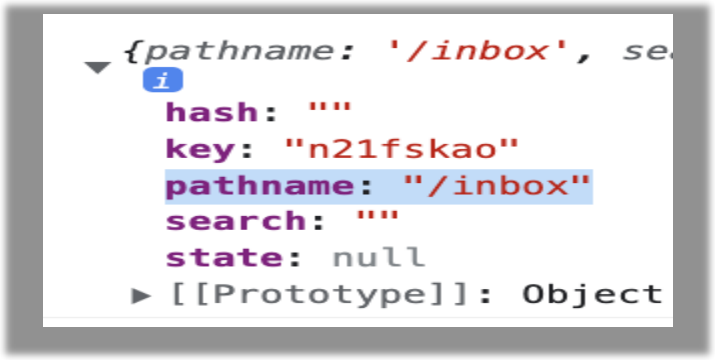
- **Solution elements: The AuthContext.**

```tsx
export const AuthContext = createContext<AuthContextInterface | null>(null);

const AuthContextProvider:React.FC<React.PropsWithChildren> = (props) => {
  const [token, setToken] = useState<string|null>(null);
  const location = useLocation();
  const navigate = useNavigate();

  const authenticate = async (username: string, password: string) => {
    const token = await fakeAuth(username, password);
    setToken(token);
    const origin = location.state?.intent?.pathname || "/";
    navigate(origin);
  };

  const signout = () => {···
  };

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

Implementation

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

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




```
5  const ProtectedRoute:React.FC<React.PropsWithChildren> = (props) => {
6      const authContext = useContext(AuthContext);
7      const { token } = authContext || {};
8      const location = useLocation();
9      if (!token) {
10         return <Navigate to={"/login"} replace state={{ intent: location }} />;
11     }
12
13     return props.children;
14 };
15
16 export default ProtectedRoute;
```

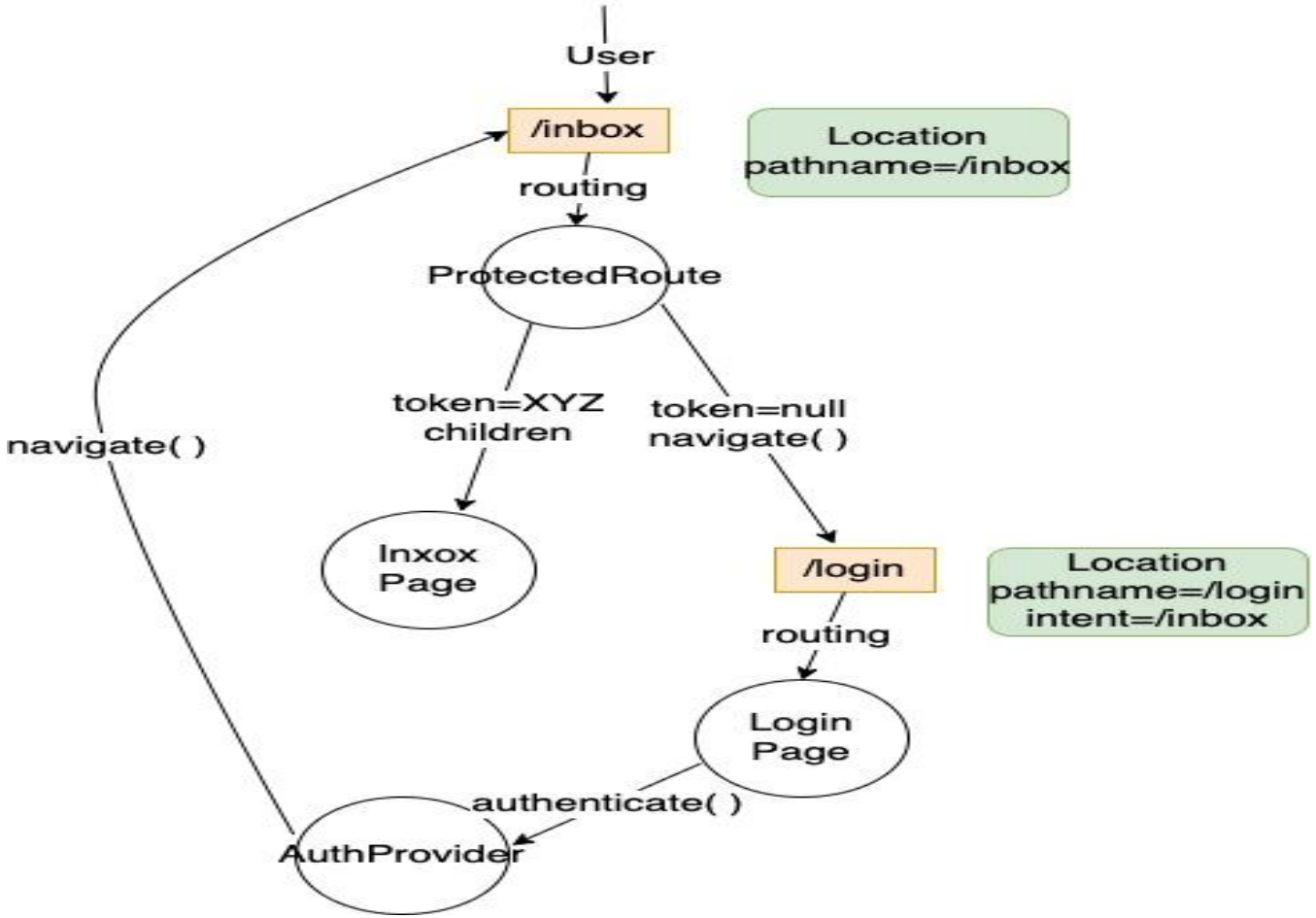
Implementation

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

```
import { useContext } from "react";
import { AuthContext } from "./authContext";

const LoginPage = () => {
  const authContext = useContext(AuthContext);
  const { authenticate } = authContext || {};

  const login = () => {
    const password = Math.random().toString(36).substring(7);
    authenticate && authenticate('user1', password);
  };

  return (
    <>
      <h2>Login page</h2>
      <p>You must log in to view the protected pages </p>
      {/* Login web form  */}
      <button onClick={login}>Submit</button>
    </>
    )
};

export default LoginPage;
```

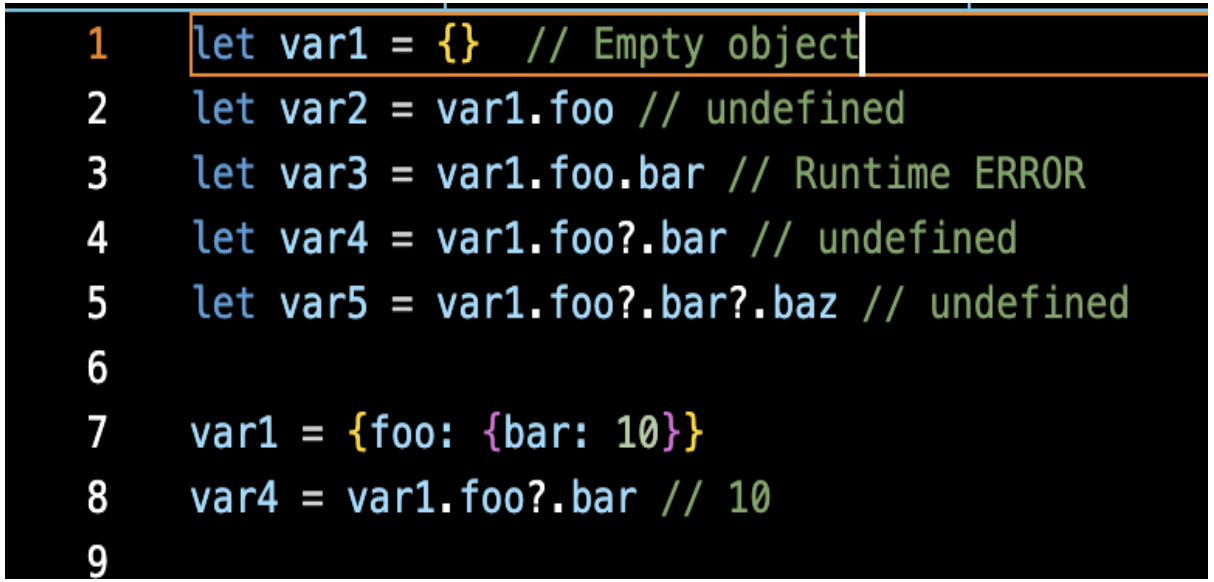
Implementation - Flow of control.

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.**