7 - React (Context API)

React education, 2024.

MITHRIL

Overview

- Context API
- Cookies
- Web storage

Context API

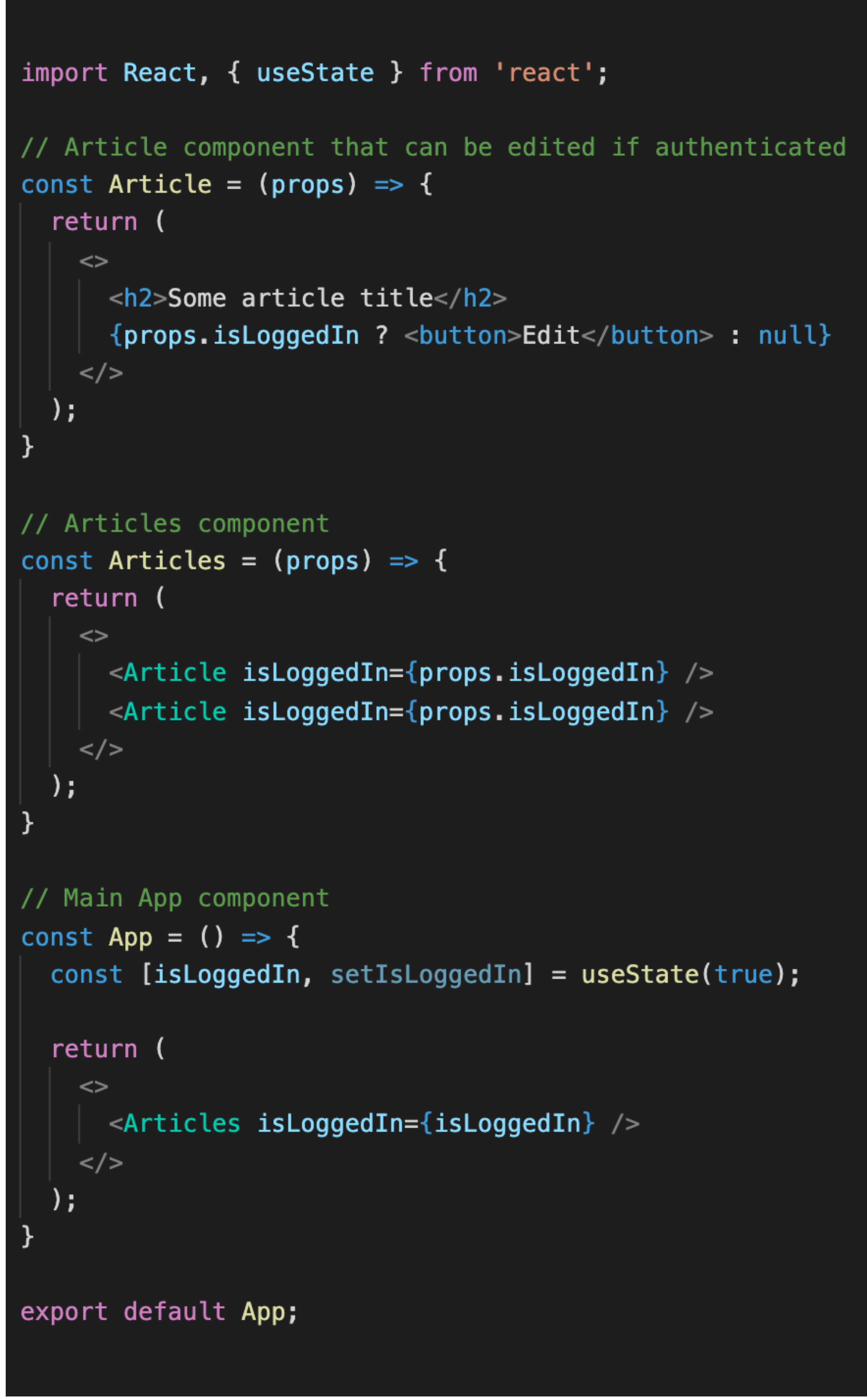
Context API

- Context provides a way to pass data through the component tree without having to pass props down manually at every level.
- Passing data by use of props can be cumbersome if many components needs to consume passed data at different nesting levels.
- In previous lectures we've already learned how to pass data through the components by use of props and callback functions in top-down and down-top direction.
- Context provides a way to share values between components without explicitly pass a prop through every level of the three.

MITHRIL

Context API

- Example with props
- In the scenario right, in each Article we want to check if user is logged in, if it is then enable editing for that Article. Let's say that our application knows if user is really logged in only in App component.
- In that case we need to pass `isLoggedIn` state from App to Articles component, then from Articles to Article component, and finally we can check if we can edit that Article or not.



```
import React, { useState } from 'react';

// Article component that can be edited if authenticated
const Article = (props) => {
  return (
    <>
      <h2>Some article title</h2>
      {props.isLoggedIn ? <button>Edit</button> : null}
    </>
  );
}

// Articles component
const Articles = (props) => {
  return (
    <>
      <Article isLoggedIn={props.isLoggedIn} />
      <Article isLoggedIn={props.isLoggedIn} />
    </>
  );
}

// Main App component
const App = () => {
  const [isLoggedIn, setIsLoggedIn] = useState(true);

  return (
    <>
      <Articles isLoggedIn={isLoggedIn} />
    </>
  );
}

export default App;
```

Context API

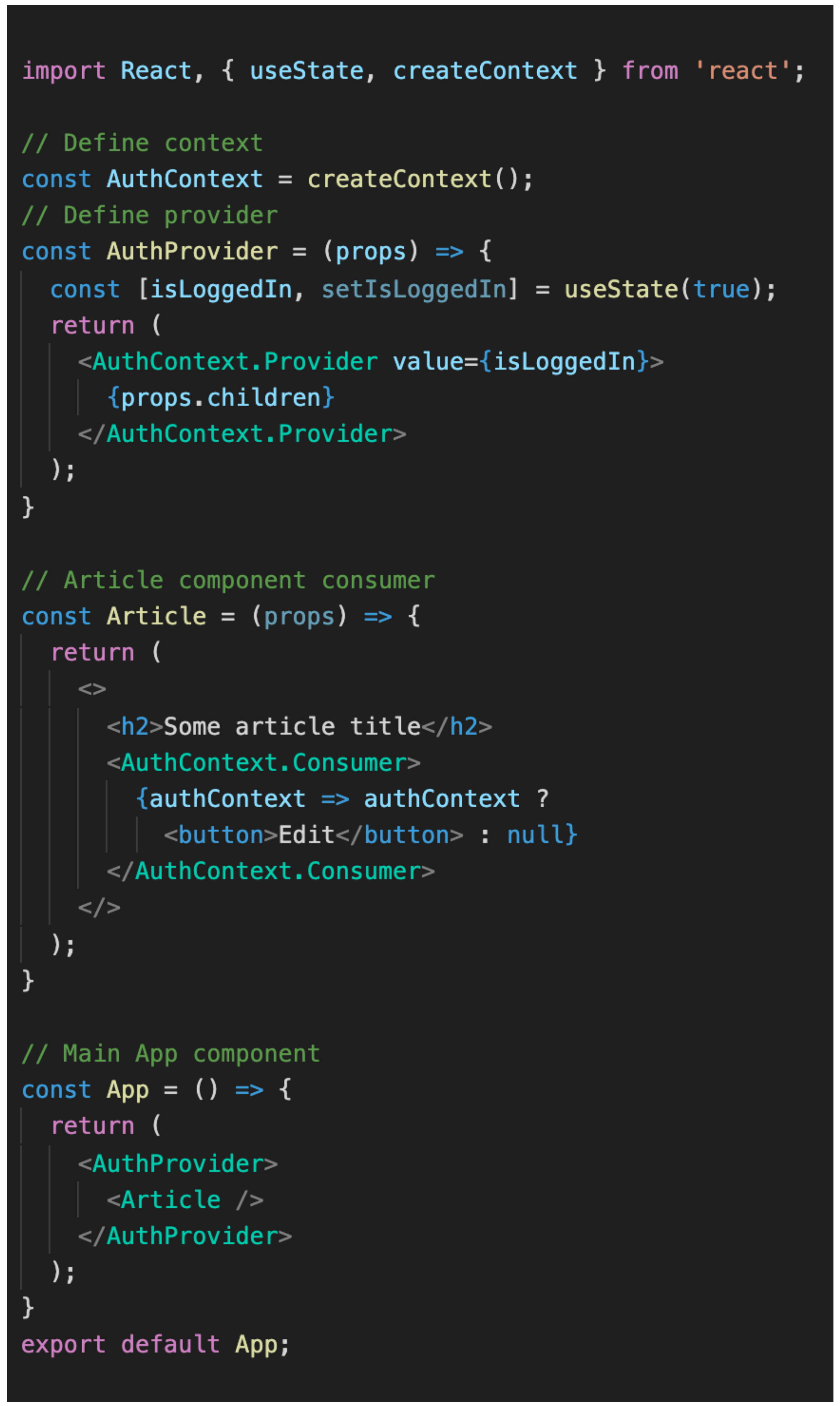
- Example with Context API
- Same outcome is achieved here, but on this way each component that is child of Provider (direct or not) has access to defined context.
- For the purpose of this example we’ve defined everything inside the same file.

```
import React, { useState, createContext } from 'react';

// Define context
const AuthContext = createContext();
// Define provider
const AuthProvider = (props) => {
  const [isLoggedIn, setIsLoggedIn] = useState(true);
  return (
    <AuthContext.Provider value={isLoggedIn}>
      {props.children}
    </AuthContext.Provider>
  );
}

// Article component consumer
const Article = (props) => {
  return (
    <>
      <h2>Some article title</h2>
      <AuthContext.Consumer>
        {authContext => authContext ?
          <button>Edit</button> : null}
      </AuthContext.Consumer>
    </>
  );
}

// Main App component
const App = () => {
  return (
    <AuthProvider>
      <Article />
    </AuthProvider>
  );
}
export default App;
```

Context API

- In order to consume data passed from Provider, we need a Consumer component that subscribes to context changes.
- Consumer component requires a function as a child which receives the current context value and returns a React node.
- For In case where we want to simplify consuming context value, `useContext()` function can be used instead.

```
const isLoggedIn = useContext(AuthContext)
```

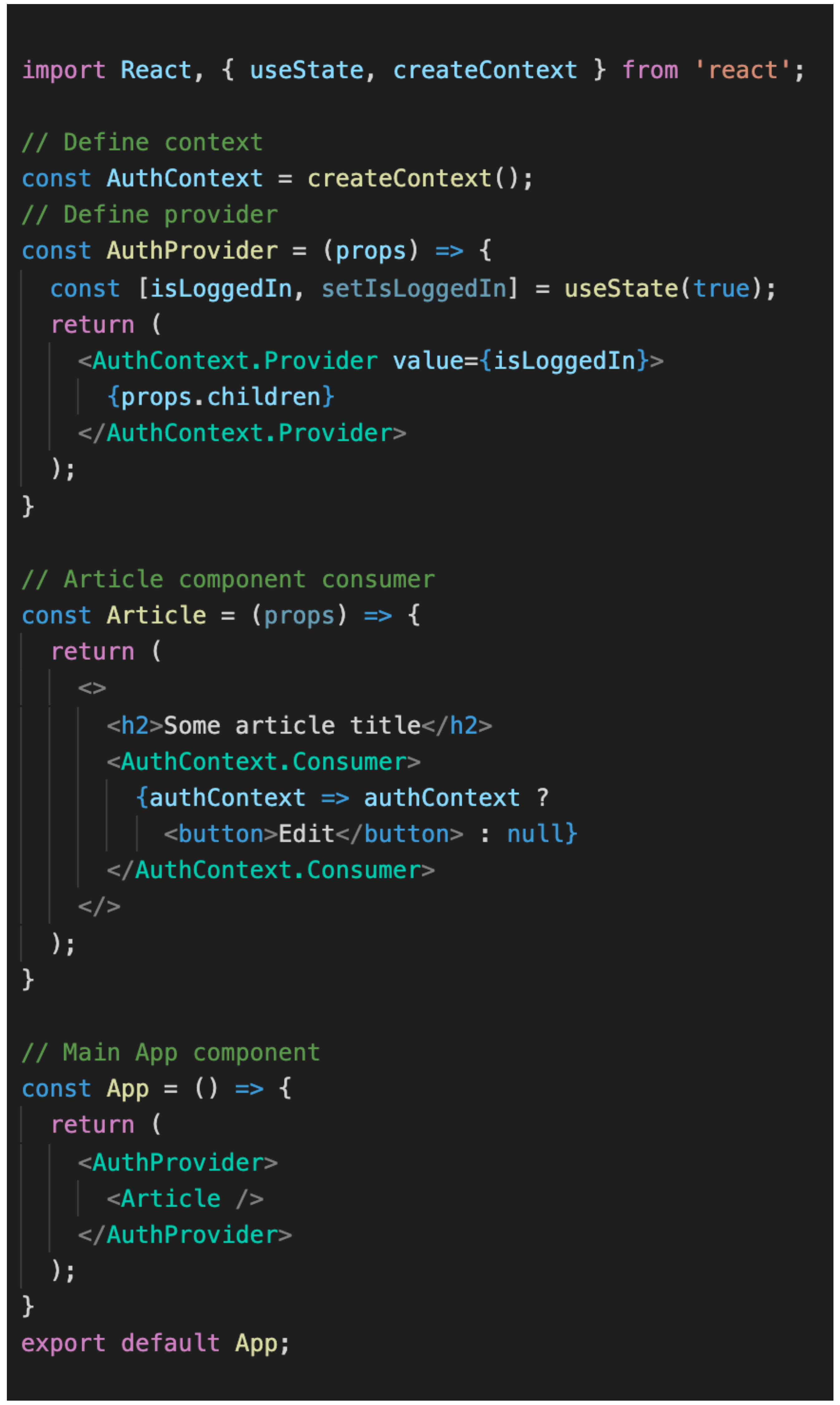


```
import React, { useState, createContext } from 'react';

// Define context
const AuthContext = createContext();
// Define provider
const AuthProvider = (props) => {
  const [isLoggedIn, setIsLoggedIn] = useState(true);
  return (
    <AuthContext.Provider value={isLoggedIn}>
      {props.children}
    </AuthContext.Provider>
  );
}

// Article component consumer
const Article = (props) => {
  return (
    <>
      <h2>Some article title</h2>
      <AuthContext.Consumer>
        {authContext => authContext ?
          <button>Edit</button> : null}
      </AuthContext.Consumer>
    </>
  );
}

// Main App component
const App = () => {
  return (
    <AuthProvider>
      <Article />
    </AuthProvider>
  );
}
export default App;
```

MITHRIL

Context API

- Solves problem when we don’t have parent-child relationship between components.
- Context is designed to share data that can be considered “global” for a tree of React components.
- Avoid using Context API for state management, but just for simpler tasks:
  - Authenticated users
  - Theme options
  - Preferred language



```
import React, { useState, createContext } from 'react';

// Define context
const AuthContext = createContext();
// Define provider
const AuthProvider = (props) => {
  const [isLoggedIn, setIsLoggedIn] = useState(true);
  return (
    <AuthContext.Provider value={isLoggedIn}>
      {props.children}
    </AuthContext.Provider>
  );
}

// Article component consumer
const Article = (props) => {
  return (
    <>
      <h2>Some article title</h2>
      <AuthContext.Consumer>
        {authContext => authContext ?
          <button>Edit</button> : null}
      </AuthContext.Consumer>
    </>
  );
}

// Main App component
const App = () => {
  return (
    <AuthProvider>
      <Article />
    </AuthProvider>
  );
}
export default App;
```

MITHRIL

Hands-on

Hands-on

- Try to create your own Context, for example to share across the app an information if the user is logged in or not. If user is not logged in, disable the button which we use for adding a new Todo.

MITHRIL

Cookies

Cookies

- Cookies were invented in 1994 by a guy named Louis J. Montulli, a software engineer at age 24, who worked at Netscape at the time.
- There was no way to “remember” what items users put into their shopping carts when they navigated between different pages of the website.
- Louis heard the term “Magic Cookie” from an operating systems courses in college to describe an opaque piece of data held by an intermediary, so he named his patent the Cookie.
- Check more on this story here.

MITHRIL

Cookies

- Cookies are used for client-side data storage:
  - Small text files stored on a user’s computer (included in each HTTP request to server).
  - Mostly used for user activity tracking, session management (logins, shopping carts) and personalization (user preferences, themes).
  - With respect to their lifetime, cookies can be divided into **session** (expiry on browser close) and **permanent** cookies (expiry on specified date/time).
  - Cookies are visible and editable from your browser Dev tools.
  - Today modern API’s like Web Storage (localStorage or sessionStorage) are used for that purpose. Cookies, localStorage & sessionStorage has some specific advantages & disadvantages, so understanding of each is required.

MITHRIL

Cookies

- In most cases where we use RESTful web services and API calls to some REST API, Cookies will be used to store JWT token.
- Cookies, when used with the HttpOnly cookie flag, are not accessible through JavaScript, and are immune to XSS.
- You can also set the Secure cookie flag to guarantee the cookie is only sent over HTTPS
- Most of CSRF can be prevented with the `SameSite` in addition with `HttpOnly` and `Secure` flags.

MITHRIL

Web storage

Web storage

- Web storage API can store data locally within the user's browser, and provides two objects for storing data on the client:
  - `localStorage` - stores data with no expiration date
  - `sessionStorage` - stores data for one session (data is lost when browser tab is closed)
- localStorage is accessible through JS on the same domain. This means that any JS running on your site will have access to web storage, and because of this can be vulnerable to XSS.
- If only one of JS script is compromised, malicious JS can be embedded on the page, and Web Storage is compromised.
- Bunch of organizations advise not to store anything of value of trust in the Web Storage.
- Web Storage doesn't enforce any secure standards during transfer, you must ensure they always send JWT over HTTPS and never HTTP.

MITHRIL

Web storage

- Both localStorage and sessionStorage poses Storage object methods:
  - localStorage.setItem(); sessionStorage.setItem()
  - localStorage.getItem(); sessionStorage.getItem()
  - localStorage.removeItem(); sessionStorage.removeItem()
  - localStorage.clear(); sessionStorage.clear()

MITHRIL

Hands-on

Hands-on

- Currently when you refresh the browser all the TODOs disappear. Use web storage to store TODOs and make them persistent.