Redux V.S. Context API

WARRANT TSAI

Web Dev

State Management Tool

State management is a critical aspect of React development, and choosing the right tool can significantly impact the efficiency and scalability of an application.

In React, components have a built-in state object where persistent data is stored between component renderings. As users interact with the application, changing the state triggers updates in the UI, reflecting the new state of the application. State management is essential for handling complex applications where multiple components need to share and respond to changes in data.

Two prominent state management tools in React are **Context API** and **Redux**, each offering distinct features and benefits.

Warrant TSAI **March 4, 2024**

Context API

Context API is a built-in feature of React that facilitates data transfer across components, reducing the need for prop drilling.

- **Goods**: Easy to use, lightweight, flexible for sharing any type of state between components efficiently.

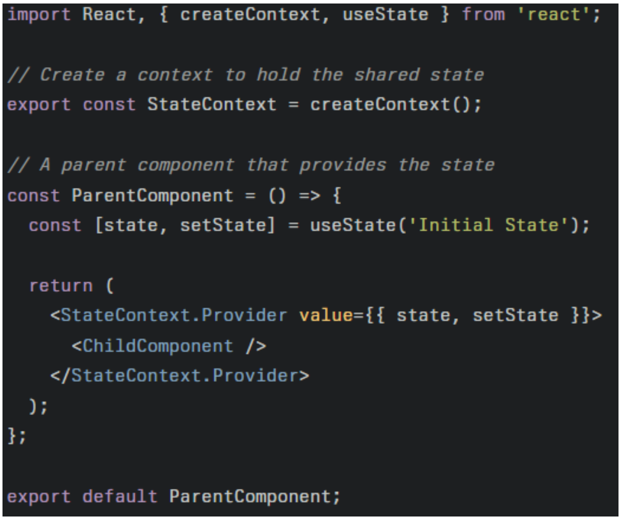
- **Bads**: Challenging in large applications due to difficulties in managing multiple contexts. For example, you can't pass the state from the child component back to the parent component. That means, you have to move the state to the parent component and then use Context API to pass to the children.

Context API Code Example (1/2)

In this example, you can observe the practical implementation of the Context API in a real project.

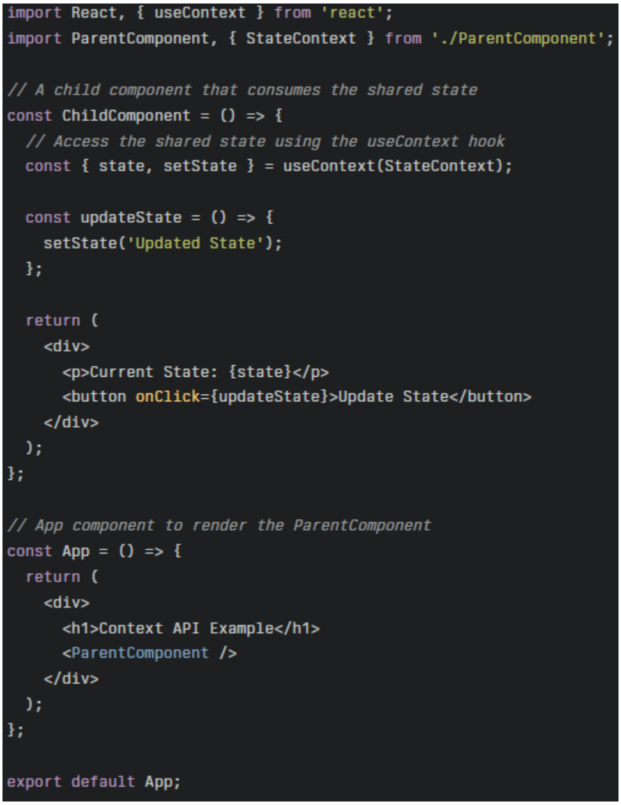
Typically, the state is defined in the parent component, and the Context API is utilized to establish a state context provider.

Subsequently, the children components can access this state using the useContext() hook.



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

// Create a context to hold the shared state
export const StateContext = createContext();

// A parent component that provides the state
const ParentComponent = () => {
  const [state, setState] = useState('Initial State');

  return (
    <StateContext.Provider value={{ state, setState }}>
      <ChildComponent />
    </StateContext.Provider>
  );
};

export default ParentComponent;
```

Context API Code Example (2/2)

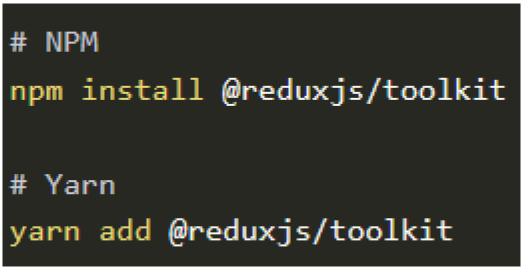


```
import React, { useContext } from 'react';
import ParentComponent, { StateContext } from './ParentComponent';

// A child component that consumes the shared state
const ChildComponent = () => {
  // Access the shared state using the useContext hook
  const { state, setState } = useContext(StateContext);

  const updateState = () => {
    setState('Updated State');
  };

  return (
    <div>
      <p>Current State: {state}</p>
      <button onClick={updateState}>Update State</button>
    </div>
  );
};

// App component to render the ParentComponent
const App = () => {
  return (
    <div>
      <h1>Context API Example</h1>
      <ParentComponent />
    </div>
  );
};

export default App;
```

REDUX (1/2)

Official website: *https://redux.js.org*

Redux is a powerful and predictable state container designed for JavaScript applications. It enables developers to build applications that exhibit consistent behavior, run seamlessly across various environments (client, server, and native), and are easily testable. Redux serves as a centralized tool for managing an application's state and logic, offering capabilities like undo/redo functionality, state persistence, and more.



```
# NPM
npm install @reduxjs/toolkit

# Yarn
yarn add @reduxjs/toolkit
```

REDUX (2/2)

- **Advantages**: Offers structured state management, predictable updates, centralized state accessible across components.

- **Disadvantages**: Steeper learning curve, setup overhead for smaller projects, more suited for global state management.

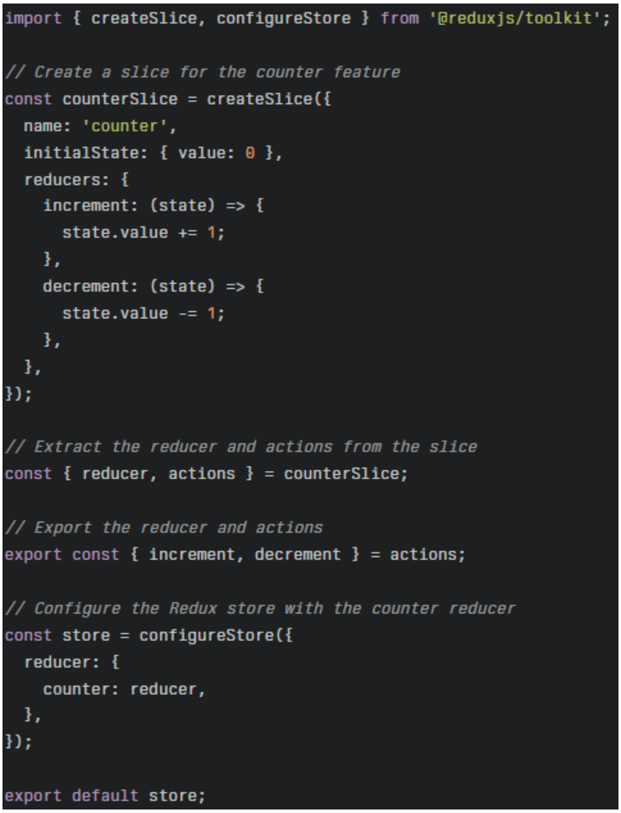
Redux Architecture:

1. **State Management**: Redux manages the application state in a single store.
2. **Reducers**: Functions that specify how the state changes in response to actions.
3. **Slices**: Collections of reducer logic and actions for specific features in the app.

Web Dev

REDUX Code Example (1/2)

Create a slice:



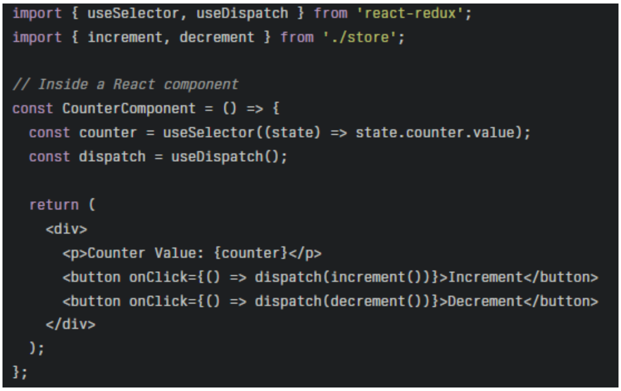
```javascript
import { createSlice, configureStore } from '@reduxjs/toolkit';

// Create a slice for the counter feature
const counterSlice = createSlice({
  name: 'counter',
  initialState: { value: 0 },
  reducers: {
    increment: (state) => {
      state.value += 1;
    },
    decrement: (state) => {
      state.value -= 1;
    },
  },
});

// Extract the reducer and actions from the slice
const { reducer, actions } = counterSlice;

// Export the reducer and actions
export const { increment, decrement } = actions;

// Configure the Redux store with the counter reducer
const store = configureStore({
  reducer: {
    counter: reducer,
  },
});

export default store;
```

Warrant TSAI **March 4, 2024**

Web Dev

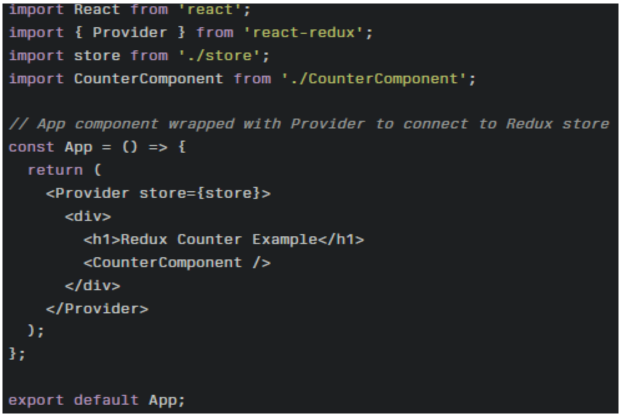
REDUX Code Example (2/2)

Connect component to REDUX store:



```
import { useSelector, useDispatch } from 'react-redux';
import { increment, decrement } from './store';

// Inside a React component
const CounterComponent = () => {
  const counter = useSelector((state) => state.counter.value);
  const dispatch = useDispatch();

  return (
    <div>
      <p>Counter Value: {counter}</p>
      <button onClick={() => dispatch(increment())}>Increment</button>
      <button onClick={() => dispatch(decrement())}>Decrement</button>
    </div>
  );
};
```

Integrating Redux in App Component:



```
import React from 'react';
import { Provider } from 'react-redux';
import store from './store';
import CounterComponent from './CounterComponent';

// App component wrapped with Provider to connect to Redux store
const App = () => {
  return (
    <Provider store={store}>
      <div>
        <h1>Redux Counter Example</h1>
        <CounterComponent />
      </div>
    </Provider>
  );
};

export default App;
```

Warrant TSAI **March 4, 2024**

Web Dev

Summary

State management is a crucial aspect of React development, ensuring efficient data handling and UI updates. Two primary tools for state management in React are Context API and Redux, each with distinct features and use cases.

Context API is ideal for simpler applications requiring easy data sharing, while Redux excels in managing complex applications with structured state requirements. Choosing between them depends on the project's scale and the need for centralized or distributed state management.

Warrant TSAI **March 4, 2024**