4_REDUX - 2

Agenda

- Revision
- Redux Toolkit - (data flow)
- Redux - (async task)
- Redux Dev Tools

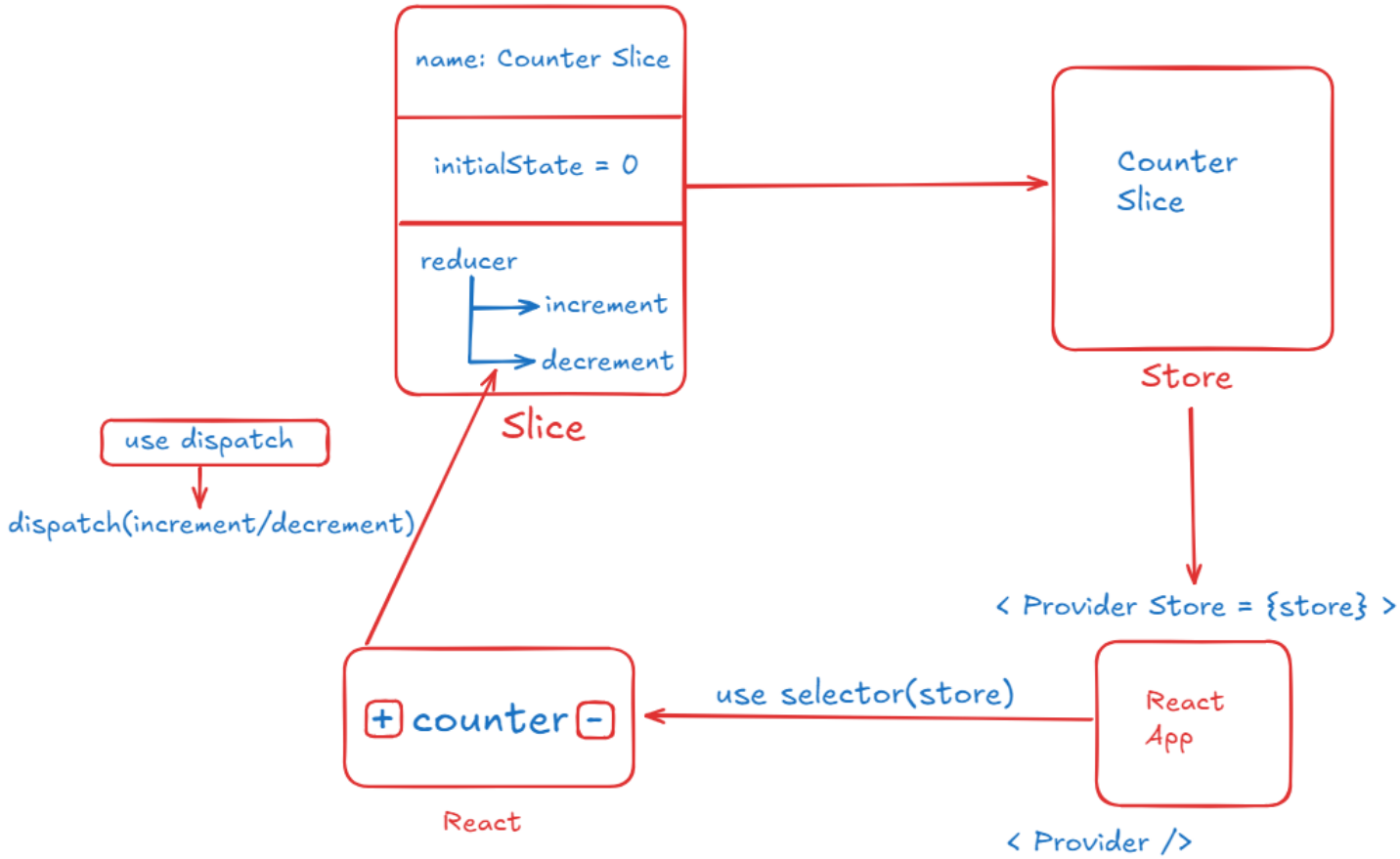
Every component consists of three parts:

- UI
- State Management
- Business Logic

Redux is a powerful state management library that excels at centralizing and managing application state in a predictable and efficient way. State management is done in two ways:

- Set
- Update

To summarize, Redux is a valuable tool for managing application state, especially in large-scale projects with complex state flows.



Work Flow

1. Slices:

- **Name:** A unique identifier for the slice.
- **InitialState:** The initial state of the slice.
- **Reducer:** A pure function that takes the current state and an action as input and returns a new state.

2. Store:

- **Creation:** The store is created by combining multiple slices.
- **Purpose:** Holds the entire application state and provides a way to access and update it.

3. Provider:

- **Wrapping:** The `Provider` component is used to wrap the entire React application.
- **Store Passing:** The store is passed as a prop to the `Provider`.

4. Accessing State:

- `useSelector` **Hook:** This hook is used to select specific parts of the state from the store.
- **State Access:** The selected state can then be used in components to render UI or perform logic.

5. Updating State:

- `useDispatch` **Hook:** This hook provides a dispatch function.
- **Dispatching Actions:** Actions are dispatched to the store, triggering updates in the reducers.

Principles

- In redux you have only one store.
- Redux has unidirectional flow of data.
- State is read only.
- You have to update through dispatch.

NormalInputCounter.jsx

Let's create a counter where there is simply an `input box` and a `button` and the button will give you `Delta` and below this, there is (+), (-), and number. We are making three sets of variables:

- Count = 0
- Delta
- Value

```
import { useState } from "react";

function NormalInputCounter() {
    //  state management  -> set
    const [count, setCount] = useState(0);
    const [delta, setDelta] = useState(1);
    const [value, setValue] = useState("");

    // bussiness logic
```

```
    const increment = () => {
        setCount(count + delta);
    }
    const decrement = () => {
        setCount(count - delta);
    }
    const updateDeltaHandler = () => {
        setDelta(Number(value));
    }

    //  ui
    return (<>
        <div>
            <input type="number" value={value} onChange={(e) =>
setValue(e.target.value)} />
            <button onClick={updateDeltaHandler}>update Delta</button>
        </div>

        <div style={{
            height: "100vh",
            display: "flex",
            justifyContent: "center",
            alignItems: "center",
            width: "100vw"
        }}>

            <button onClick={increment}>+</button>
            <p>Count :{count}</p>
            <button onClick={decrement}>-</button>
        </div>
    </>

    )
}

export default NormalInputCounter;
```

App.jsx

```
import NormalInputCounter from "./Components/NormalInputCounter.jsx";

function App(){
        return(
```

```
                <>
                        <NormalInputCounter/>
                </>
        )
}

export default App;
```

Summary

Normally it is increased or decreased but when we put the number into the input box (for ex: 10) and click on the update Delta button then it will increase or decrease by 10.

Handling state with redux-toolkit(sending data)

Let's convert into Redux through which we learn how these things work in the Redux case.

UI will be the same there will be no changes.

CounterInputSlice.js

```
import { createSlice } from "@reduxjs/toolkit";
const counterInputSlice = createSlice({
    name: "counterInputSlice",
    initialState: {
        count: 0,
        delta: 1
    },
    // define all the possible bussiness logic
    reducers: {
        increment: (state) => {
            state.count += state.delta;

        },
        decrement: (state) => {
            state.count -= state.delta;

        },
        updateDelta: (state, params) => {
            // to acccess just params.payload
            const delta = params.payload;
            state.delta = delta;
```

```
        }
    }
})
export default counterInputSlice;
```

Store.js

We created a store once and depending upon its use case we are adding multiple slices to it.

```
import { configureStore } from "@reduxjs/toolkit";
import counterSlice from "./slice/CounterSlice.js";
import counterInputSlice from "./slice/CounterInputSlice.js";

const store = configureStore({
    reducer: {
        counterSection: counterSlice.reducer,
        counterInputSlice: counterInputSlice.reducer,
    },
})

export default store;
```

ReduxInputCounter.jsx

```
import { useState } from "react";
import { useSelector } from "react-redux";
import { useDispatch } from "react-redux";

import counterInputSlice from "../redux/slice/CounterInputSlice";
const actions = counterInputSlice.actions;

function ReduxInputContainer() {
    const [value, setValue] = useState("");

    const dispatch = useDispatch();

    //  state management  -> (slice)
    const { count, delta } = useSelector((store) => store.counterInputSlice);

    // bussiness logic
    const increment = () => {
        dispatch(actions.increment())
```

```
    }
    const decrement = () => {
        dispatch(actions.decrement())
    }
    const updateDeltaHandler = () => {
        dispatch(actions.updateDelta(value))
    }
    //  ui
    return (<>
        <div>
            <input type="number" value={value} onChange={(e) =>
setValue(e.target.value)} />
            <button onClick={updateDeltaHandler}>update Delta</button>
        </div>

        <div style={{
            height: "100vh",
            display: "flex",
            justifyContent: "center",
            alignItems: "center",
            width: "100vw"

        }}>
            <button onClick={increment}>+</button>
            <p>Count :{count}</p>
            <button onClick={decrement}>-</button>
        </div>
    </>
    )


}
export default ReduxInputContainer;
```

- For any function you call in `dispatch`, you will pass a `value` in it, which will always come under the `second parameter` of the `payload property`.

Async task

User.jsx

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

function UserComponent() {
    // loading
    // user
    //error
    const [user, setUser] = useState(null);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);
    useEffect(() => {
        async function getUser() {
            try {
                // fetch user data from API and set it to the state
                const userResp = await
fetch('https://jsonplaceholder.typicode.com/users/1');
                const userData = await userResp.json();
                setUser(userData);
            } catch (err) {
                setError(err);
            } finally {
                setLoading(false);
            }

        }
        getUser();
    }, [])

    if (loading) {
        return <div>Loading...</div>
    }
    if (error) {
        return <div>Error: {error.message}</div>
    }
    return (
        <div style={{
            display: "flex",
            justifyContent: "center",
            alignItems: "center",
            width: "100vw",

        }}>
            <div>
```

```
                <h4>Name: {user.name}</h4>
                <h4>Phone: {user.phone}</h4>
            </div>
        </div>
    )
}

export default UserComponent;
```

App.jsx

```
import UserComponent from "./Components/User.jsx";

function App(){
        return(
                <>
                <UserComponent/>
                </>
        )
}

export default App;
```

Output

When you reload the app first `Loading...` will come and after that `data` will be visible on your screen.
Data fetched successfully the user data is displayed with name and phone number.

Handling async tasks with redux

UserSlice.js

```
import { createSlice } from "@reduxjs/toolkit";
const userSlice = createSlice({
    name: "userSlice",
    initialState: {
        user: null,
        loading: true,
        error: null,
    },
    reducers: {
```

```
        onPending: (state) => {
            state.user = null;
            state.loading = true;
            state.error = null
        },
        onRejected: (state, params) => {
            state.user = null;
            state.loading = false;
            state.error = params.payload
        }
        ,
        onFulfilled: (state, params) => {
            state.user = params.payload;
            state.loading = false;
            state.error = null
        }
    }
})

export default userSlice;
```

store.js

```
import { configureStore } from "@reduxjs/toolkit";
import counterSlice from "./slice/CounterSlice.js";
import counterInputSlice from "./slice/CounterInputSlice.js";
import userSlice from "./slice/UserSlice.js"
const store = configureStore({
    reducer: {
        counterSection: counterSlice.reducer,
        counterInputSlice: counterInputSlice.reducer,
        userSection: userSlice.reducer,
    },
})

export default store;
```

Note : We will continue the async redux in the next session