Redux – State Management

Relevel
by Unacademy

Topics Covered

Introduction
Redux recap

Why do we need Redux?
What is the use of Redux

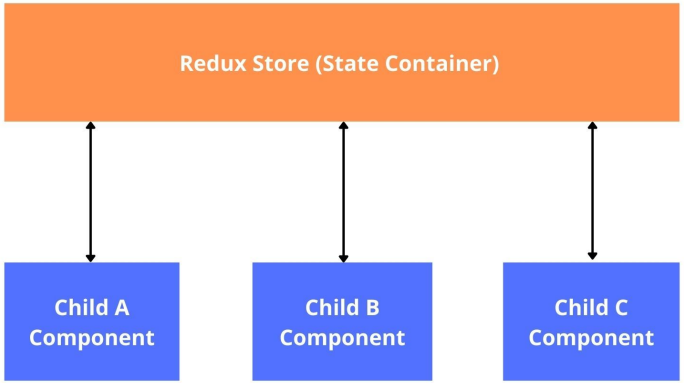
Principles of Redux
What are the three principles that we need to learn?

Redux Form
We will be creating a form bootstrapped from redux-forms

Practice Questions
Assignments

What is Redux?

- Redux is a predictable state container which can be used to communicate between the components easily.
- It helps to manage the state of an application in a redux store which can accessed by other components.



Why do we need Redux?

- In a simple react application, we use props to pass the data down to child component
- In a complex react application, they might be multiple component which want to communicate with each other. In this case props will not help manage the state properly
- To overcome this situation, Redux comes into picture.
- It seamlessly manage the state of the application by storing the data of all the component in a single store.
- This store can used by other components to get the data of some other component.

Principles of Redux

Redux has primarily three principles:

- It is the single source of truth.
  - It is the single source of truth, which means all of the application's states should be maintained and accessed through the Redux store only. This principle ensures Redux is reliable since it is the only place where state updation can take place.
- The states are read-only.
- All changes can be done through pure functions only.

Principles of Redux

Redux has primarily three principles:

- It is the single source of truth.
- The states are read-only.
    - State is Read only. State cannot be mutated/changed via API or network calls, the only way to update state is through actions. The objects dispatched via UI components are the only means to update the state, any other API call or function call without the dispatched actions won't be able to update the state in store.
- All changes can be done through pure functions only.

Principles of Redux

Redux has primarily three principles:

- It is the single source of truth.
- The states are read-only.
- All changes can be done through pure functions only.
    - All changes are made with pure functions. A pure function is a function that is bound to give the same results if the same parameters are passed. Reducers are functions that take in the previous state and action and return the next state, thus they are pure functions.

How to implement redux in react application

Create a store

```
import { configureStore } from "@reduxjs/toolkit";
import userReducer from "./userReducer";
import productReducer from "./productReducer";

export default configureStore({
  reducer: {
    user: userReducer,
    product: productReducer
  }
});
```

How to implement redux in react application

Create a reducer function

```
import { createSlice } from "@reduxjs/toolkit";

export const userReducer = createSlice({
  name: "user",
  initialState: [],
  reducers: {
    addUserDetail: (state, action) => {
      state = [...state, action.payload];
      return state;
    }
  }
});
```

How to implement redux in react application

Dispatch the action with payload

```
let payload = {
  name: userName,
  email: userEmail
};

dispatch(addUserDetail(payload));
```

Read the store data using useSelector redux hook

```
const userList = useSelector((state) => state.user);
```

Building the feature of today's session using the concepts learned

- A React application which will -
  - Fetch the user and product detail from one component to other using redux.
  - Display the newly added user and product in front-end.
- Link:- https://codesandbox.io/s/festive-bush-hqqtn

Practice Code

- Implement a counter application where on click ‘+’ increment counter by 1 and on click ‘-’ decrement counter by 1 and add button named ‘Increment if odd’, on click of that increment counter by 1 if the counter value is odd, one more button to increment counter Asynchronous using setTimeOut to 1sec i.e 1000 milliseconds

Practice Code Solution

Code Link: https://jsfiddle.net/3azy2exs/2/

Practice Code Solution

HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Redux basic example</title>
    <script
src="https://unpkg.com/redux@latest/dist/redux.min.js">
</script>
  </head>
  <body>
    <div>
      <p>
        Clicked: <span id="value">0</span> times
        <button id="increment">+</button>
```

Practice Code Solution

HTML:

```
        <button id="decrement">-</button>
        <button id="incrementIfOdd">Increment if
odd</button>
        <button id="incrementAsync">Increment
async</button>
      </p>
    </div>
  </body>
</html>
```

Practice Code Solution

JavaScript:

```
function counter(state, action) {
        if (typeof state === 'undefined') {
          return 0
        }
        switch (action.type) {
          case 'INCREMENT':
            return state + 1
          case 'DECREMENT':
            return state - 1
          default:
            return state
        }
      }
```

Practice Code Solution

JavaScript:

```
      var store = Redux.createStore(counter)
      var valueEl = document.getElementById('value')
function render() {
        valueEl.innerHTML = store.getState().toString()
      }

      render()
      store.subscribe(render)
      document.getElementById('increment')
        .addEventListener('click', function () {
          store.dispatch({ type: 'INCREMENT' })
        })
```

Practice Code Solution

JavaScript:

```
    document.getElementById('decrement')
      .addEventListener('click', function () {
        store.dispatch({ type: 'DECREMENT' })
      })

document.getElementById('incrementIfOdd')
      .addEventListener('click', function () {
        if (store.getState() % 2 !== 0) {
          store.dispatch({ type: 'INCREMENT' })
        }
      })
```

Practice Code Solution

JavaScript:

```
document.getElementById('incrementAsync')
  .addEventListener('click', function () {
    setTimeout(function () {
      store.dispatch({ type: 'INCREMENT' })
    }, 1000)
  })
```

Thank you!