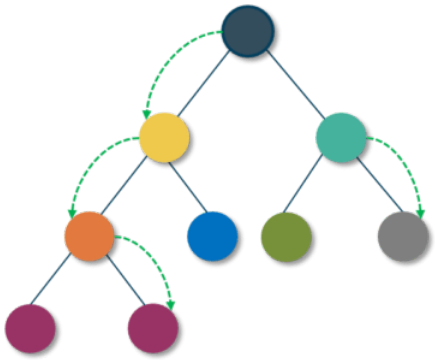
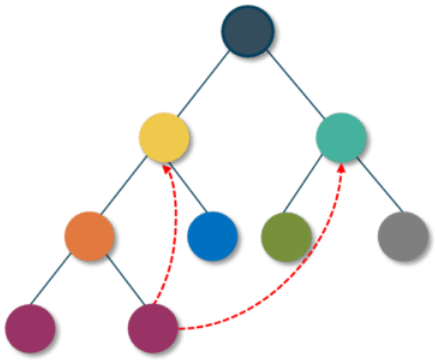
21. React Redux

Table of Content

1. Introduction to Redux with React





Redux is a state management library. React follows the component-based approach, where the data flows through the components. In fact, the data in React always flows from parent to child components which makes it *unidirectional*

But what happens when we try to communicate from a non-parent component?

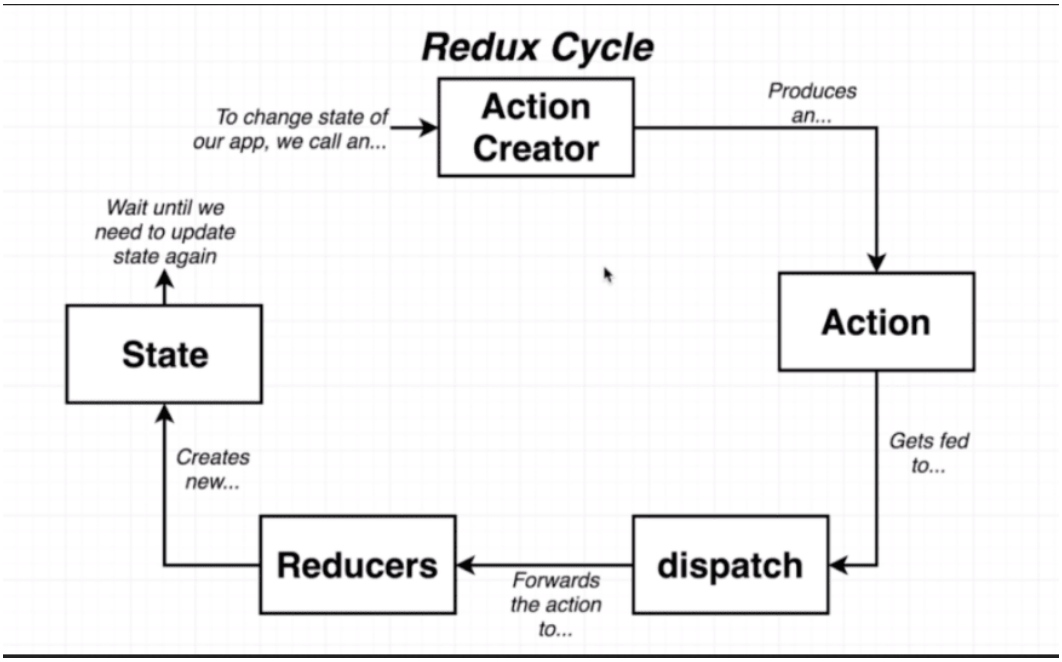
A child component can never pass data back up to the parent component. React does not provide any way for direct component-to-component communication. Even though React has features to support this approach, it is considered to be a poor practice. It is prone to errors and leads to spaghetti code. So, how can two non-parent components pass data to each other?

This is where React fails to provide a solution and **Redux comes into the picture.**

Redux provides a “**store**” as a solution to this problem. A store is a place where you can store all your application state together. Now the components can “*dispatch*” state changes to the store and not directly to the other components. Then the components that need the updates about the state changes can “*subscribe*” to the store.

Thus, with Redux, it becomes clear where the components get their state from as well as where should they send their states to. Now the component initiating the change does not have to worry about the list of components needing the state change and can simply dispatch the change to the store. This is how Redux makes the *data flow* easier.

2. How Redux works



ACTION: An object setting up information about our data moving into state.

DISPATCH: Functions which act as a bridge between our actions and reducers sending the information over to our application.

REDUCER: A function that receives data information from our action in the form of a type and payload and modifies the state based on additional provided data.

STATE: Where all the data from our reducers are passed into a central source.

Store Once we have our Actions and Reducers set up everything is maintained in our Store. The Store is where we can set up our initial state, combine our reducers, apply middleware and hold our information.

Provider We can then use the Provider to wrap our store around our application and by doing so pass and render our state. We *can then use Connect *with a component and enabling it to receive **Store** state from the **Provider**.

3. Working implementation of Redux

▼ **Install the following packages:**

```
npm install @reduxjs/toolkit react-redux
```

▼ **Create Reducers & Actions using createSlice**

Creating a slice requires a string name to identify the slice, an initial state value, and one or more reducer functions to define how the state can be updated. Once a slice is created, we can export the generated Redux action creators and the reducer function for the whole slice.

```
import { createSlice } from '@reduxjs/toolkit'

const initialState = {
  value: 0,
}

export const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
```

```
    increment: (state) => {
      state.value += 1
    },
    decrement: (state) => {
      state.value -= 1
    },
    incrementByAmount: (state, action) => {
      state.value += action.payload
    },
  },
})

export const { increment, decrement, incrementByAmount } = counterSlice.actions

export default counterSlice.reducer
```

▼ Add Slice Reducers to the Store

A store is a JavaScript object which can hold the application’s state and provide a few helper methods to access the state, dispatch actions and register listeners. The entire state/ object tree of an application is saved in a single store. As a result of this, Redux is very simple and predictable.

```
import { configureStore } from '@reduxjs/toolkit'
import counterReducer from '../features/counter/counterSlice'

export const store = configureStore({
  reducer: {
    counter: counterReducer,
  },
})
```

▼ Add store provider to the App

```
import App from './App'
import { store } from './app/store'
import { Provider } from 'react-redux'

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
```

```
  document.getElementById('root')
)
```

▼ Dispatch actions in Components

Now we can use the React-Redux hooks to let React components interact with the Redux We can dispatch actions using `useDispatch`

```
import React from 'react'
import { useDispatch } from 'react-redux'
import { decrement, increment } from './counterSlice'

export function Counter() {

  const dispatch = useDispatch()

  return (
    <div>
      <div>
        <button
          onClick={() => dispatch(increment())}
        >
          Increment
        </button>
        <button
          onClick={() => dispatch(decrement())}
        >
          Decrement
        </button>
      </div>
    </div>
  )
}
```

▼ Read state data in components

We can read data from the store with useSelector hook.

```
import React from 'react'
import { useSelector } from 'react-redux'

export function ShowCounter() {
```

```
  const count = useSelector((state) => state.counter.value)

  return (
    <div>
      <div>

        <span>{count}</span>

      </div>
    </div>
  )
}
```

4. Advantages of Redux

Following are some of the major advantages of Redux:

- **Predictability of outcome –** Since there is always one source of truth, i.e. the store, there is no confusion about how to sync the current state with actions and other parts of the application.
- **Maintainability –** The code becomes easier to maintain with a predictable outcome and strict structure.
- **Developer tools –** From actions to state changes, developers can track everything going on in the application in real time.
- **Community and ecosystem –** Redux has a huge community behind it which makes it even more captivating to use. A large community of talented individuals contribute to the betterment of the library and develop various applications with it.
- **Ease of testing –** Redux code are mostly functions which are small, pure and isolated. This makes the code testable and independent.
- **Organization –** Redux is very precise about how the code should be organized, this makes the code more consistent and easier when a team works with it.

Assignments

1. Build a Notes App with Create, Update and Delete Notes functionalities using Redux.
2. Create a Login page and store the user information inside the redux store, display the information inside the profile page using React Redux.
3. Create a cart app, where you can create new cart item along with the counter to increment and decrement the quantities of the items. Inside the Navbar component, display the total amount of items and total cart amount using React Redux