Unit 4 : Redux

Create and configure redux store:

Creating and configuring a Redux store involves a few steps. Here's a step-by-step solution to create and configure a Redux store in a React application:

1. **Install Redux**:

   First, make sure you have Redux installed in your project. You can install it using npm or yarn:

   ```sh
npm install redux react-redux
   ```

2. **Create Reducer**:

   Create a reducer function that specifies how the state changes in response to actions. This function will define the initial state and how the state updates for different action types.

   ```javascript
   // src/reducers/counterReducer.js
   const initialState = {
     count: 0
   };

   const counterReducer = (state = initialState, action) => {
     switch (action.type) {
       case 'INCREMENT':
         return {
           ...state,
           count: state.count + 1
   ```

```
      };
    case 'DECREMENT':
      return {
        ...state,
        count: state.count - 1
      };
    default:
      return state;
  }
};

export default counterReducer;
```

3. **Combine Reducers (Optional)**:

   If your application has multiple reducers, you can combine them using the `combineReducers` function provided by Redux.

```javascript
// src/reducers/index.js
import { combineReducers } from 'redux';
import counterReducer from './counterReducer';

const rootReducer = combineReducers({
  counter: counterReducer
  // Add more reducers here
});

export default rootReducer;
```

```

4. **Create Store**:

   Create the Redux store by passing your root reducer to the `createStore` function from Redux. You can also apply middleware here if needed.

   ```javascript
   // src/store.js
   import { createStore } from 'redux';
   import rootReducer from './reducers';

   const store = createStore(rootReducer);

   export default store;
   ```

5. **Wrap App with Provider**:

   Wrap your application with the `Provider` component from the `react-redux` library. The `Provider` component makes the Redux store available to all components in the application.

   ```javascript
   // src/index.js
   import React from 'react';
   import ReactDOM from 'react-dom';
   import { Provider } from 'react-redux';
   import App from './App';
   import store from './store';

ReactDOM.render(

<Provider store={store}>

<App />

</Provider>,

document.getElementById('root')

  );

  ```

6. **Access Store in Components**:

  In your React components, you can access the Redux store using the `useSelector` hook from the `react-redux` library.

  ```javascript

  // src/components/Counter.js

  import React from 'react';

  import { useSelector, useDispatch } from 'react-redux';

  function Counter() {

    const count = useSelector(state =>state.counter.count);

    const dispatch = useDispatch();

    return (

Count: {count}

<button onClick={() => dispatch({ type: 'INCREMENT' })}>Increment</button>

<button onClick={() => dispatch({ type: 'DECREMENT' })}>Decrement</button>

    );

  }

```
export default Counter;
```

With these steps, you've created and configured a Redux store in your React application. You can now use the store to manage and share state across different components efficiently. Remember that Redux might have more advanced features and concepts you can explore as your application grows in complexity.