

# Redux

redux keypointwise



## Redux: Key Points Explained

Redux is a popular state management library for JavaScript applications, especially with React. It helps manage and centralize the application's state in a predictable way.

### 1. Single Source of Truth

- **Concept:** All the state of the application is stored in a single JavaScript object called the "store."
- **Example:** Imagine an online store where the cart items, user information, and product details are all stored in one central location. This makes it easier to access and manage the data consistently across the app.

### 2. State is Read-Only

- **Concept:** The state in Redux cannot be changed directly. Instead, you use actions to describe what happened and reducers to handle these actions.
- **Example:** If you want to add an item to the cart, you don't directly modify the cart array. Instead, you dispatch an action like `{ type: 'ADD_TO_CART', item: newItem }` and let a reducer update the state.

### 3. Changes are Made with Pure Functions (Reducers)

- **Concept:** Reducers are functions that take the current state and an action as arguments and return a new state. They must be pure, meaning they don't mutate the state or have side effects.
- **Example:** A reducer for handling cart actions might look like this:

javascript

```
function cartReducer(state = [], action) {
  switch(action.type) {
    case 'ADD_TO_CART':
      return [...state, action.item];
    default:
      return state;
  }
}
```

- **Explanation:** Here, `cartReducer` takes the current state (an array of items) and an action. If the action is `ADD_TO_CART`, it returns a new array with the new item added, without mutating the original state.

## 4. Actions

- **Concept:** Actions are plain JavaScript objects that describe what happened. They usually have a `type` property and sometimes other data.
- **Example:** An action to add an item to the cart might look like this:

javascript

```
const addItemAction = { type: 'ADD_TO_CART', item: newItem };
```

- **Explanation:** This action has a `type` to describe the action and an `item` that represents the new item being added to the cart.

## 5. Dispatching Actions

- **Concept:** To update the state, you "dispatch" actions. This is like sending a signal to the Redux store to say, "Hey, something happened; update the state!"
- **Example:** In a React component, you might have a button that dispatches an action when clicked:

javascript

```
function addToCart(item) {
  store.dispatch({ type: 'ADD_TO_CART', item });
}
```

## 6. Store

- **Concept:** The store is the object that brings everything together. It holds the application's state, allows access to the state, dispatches actions, and registers listeners via `subscribe`.
- **Example:** To create a store, you use the `createStore` function from Redux:

javascript

```
import { createStore } from 'redux';
const store = createStore(cartReducer);
```

## 7. Middleware

- **Concept:** Middleware provides a way to extend Redux with custom functionality. It's often used for logging, crash reporting, or handling asynchronous actions.
- **Example:** A popular middleware is `redux-thunk`, which allows you to write action creators that return a function instead of an action. This is useful for handling asynchronous operations like fetching data from an API.

## 8. Selectors

- **Concept:** Selectors are functions that extract specific pieces of data from the state. They help keep components decoupled from the state structure.
- **Example:** A selector to get all items in the cart might look like this:

```
javascript
```

```
const selectCartItem = (state) => state.cart;
```

## Example: Simple Shopping Cart Application

- **State Structure:**

```
javascript
```

```
{
  cart: [],
  user: { name: 'John', loggedIn: true },
  products: [...]
}
```

- **Action:** `{ type: 'ADD\_TO\_CART', item: { id: 1, name: 'Apple', price: 1.00 } }`
- **Reducer:**

```
javascript
```

```
function cartReducer(state = [], action) {
  switch(action.type) {
    case 'ADD_TO_CART':
      return [...state, action.item];
    default:
      return state;
  }
}
```

In this example, the entire state of the application, including cart items, user information, and product details, is managed in one central store. Actions are dispatched to make changes to the state, and reducers handle these actions to produce the next state.

## Summary

Redux helps manage state predictably in complex applications by providing a structured way to handle state updates and keep everything centralized. It ensures that changes to the state are explicit and traceable, which is especially valuable for debugging and maintaining applications.



made one application code with react and redux where you use every thing of above points.



To demonstrate a simple application using React and Redux, we'll create a basic shopping cart. This example will include the following:

1. **Single Source of Truth:** The entire application's state will be managed in a single Redux store.
2. **State is Read-Only:** We'll use actions to update the state.
3. **Reducers:** We'll define reducers to handle these actions and update the state.
4. **Actions:** Define actions to interact with the state.
5. **Dispatching Actions:** We'll dispatch actions from React components.
6. **Middleware:** We'll use `redux-thunk` for handling asynchronous actions.
7. **Selectors:** We'll use selectors to get data from the state.

# Setup

## 1. React and Redux Installation

You can create a new React app and install Redux-related packages using the following commands:

```
bash
```

```
npx create-react-app redux-shopping-cart
cd redux-shopping-cart
npm install redux react-redux redux-thunk
```

## File Structure

- `src`
  - `components`
    - `ProductList.js`
    - `Cart.js`
  - `redux`
    - `actions.js`
    - `reducers.js`
    - `store.js`
  - `App.js`
  - `index.js`

## 1. Store Configuration (`store.js`)

```
javascript
```

```
// src/redux/store.js
import { createStore, applyMiddleware, combineReducers } from 'redux';
import thunk from 'redux-thunk';
import cartReducer from './reducers';

const rootReducer = combineReducers({
  cart: cartReducer,
});

const store = createStore(rootReducer, applyMiddleware(thunk));

export default store;
```

## 2. Reducer (`reducers.js`)

```
javascript
```

```
// src/redux/reducers.js
const initialState = {
  items: [],
};

function cartReducer(state = initialState, action) {
  switch (action.type) {
    case 'ADD_TO_CART':
      return { ...state, items: [...state.items, action.item] };
    case 'REMOVE_FROM_CART':
      return { ...state, items: state.items.filter(item => item.id !== action.itemId) };
    default:
      return state;
  }
}

export default cartReducer;
```

### 3. Actions (`actions.js`)

javascript

```
// src/redux/actions.js
export const addToCart = (item) => {
  return {
    type: 'ADD_TO_CART',
    item,
  };
};

export const removeFromCart = (itemId) => {
  return {
    type: 'REMOVE_FROM_CART',
    itemId,
  };
};
```

### 4. Components

#### Product List (`ProductList.js`)

javascript

```
// src/components/ProductList.js
import React from 'react';
import { useDispatch } from 'react-redux';
import { addToCart } from '../redux/actions';

const ProductList = () => {
  const dispatch = useDispatch();

  const products = [
    { id: 1, name: 'Apple', price: 1.00 },
    { id: 2, name: 'Banana', price: 0.50 },
    { id: 3, name: 'Orange', price: 0.75 },
  ];

  const handleAddToCart = (product) => {
    dispatch(addToCart(product));
  };

  return (
    <div>
      <h2>Product List</h2>
      <ul>
        {products.map(product => (
          <li key={product.id}>
            {product.name} - ${product.price.toFixed(2)}
            <button onClick={() => handleAddToCart(product)}>Add to Cart</button>
          </li>
        ))}
      </ul>
    </div>
  );
};

export default ProductList;
```

#### Cart (`Cart.js`)

javascript

```
// src/components/Cart.js
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { removeFromCart } from '../redux/actions';
```

```

const Cart = () => {
  const items = useSelector((state) => state.cart.items);
  const dispatch = useDispatch();

  const handleRemoveFromCart = (itemId) => {
    dispatch(removeFromCart(itemId));
  };

  return (
    <div>
      <h2>Cart</h2>
      {items.length === 0 ? (
        <p>Your cart is empty.</p>
      ) : (
        <ul>
          {items.map(item => (
            <li key={item.id}>
              {item.name} - ${item.price.toFixed(2)}
              <button onClick={() => handleRemoveFromCart(item.id)}>Remove</button>
            </li>
          ))}
        </ul>
      )}
    </div>
  );
};

export default Cart;

```

## 5. Main App Component (`App.js`)

javascript

```

// src/App.js
import React from 'react';
import ProductList from './components/ProductList';
import Cart from './components/Cart';

function App() {
  return (
    <div className="App">
      <h1>Shopping Cart</h1>
      <ProductList />
      <Cart />
    </div>
  );
}

export default App;

```

## 6. Entry Point (`index.js`)

javascript

```

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

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
);

```

## Explanation

1. **Single Source of Truth:** The entire application state, including cart items, is managed in the Redux store.
2. **State is Read-Only:** State is updated only through dispatched actions.
3. **Reducers:** The ``cartReducer`` handles the actions to update the state.
4. **Actions:** We have ``addToCart`` and ``removeFromCart`` actions.
5. **Dispatching Actions:** Actions are dispatched from the ``ProductList`` and ``Cart`` components.
6. **Middleware:** We included ``redux-thunk`` for potential future use (not used directly in this example).
7. **Selectors:** The ``useSelector`` hook is used to access the state in the ``Cart`` component.

This application demonstrates how Redux can be used to manage the state of a React application in a predictable way.