States

Basics

Introduction

States are a **single source of truth for all the data** in your JavaScript application.
Modern web applications are represented as a complex tree of **components** that **constantly produces data called states**.
When states are decentralized, it can become difficult to understand and test.

Lexicon

NAME	DEFINITION	DESCRIPTION
STORE	Single immutable object.	This **object stores** all the **application state**.
REDUCERS	List of functions defined in a slice.	Functions that take the **old state** and an **action**, then **define the logic** that's required to **change the state**.
ACTIONS	List of Functions defined in a reducer.	Functions defined inside a reducer allowing to perform an action.
DISPATCHER	Function called when a state needs to be changed.	Function that takes an action and data payload, then dispatch them to the store.

Redux

Redux is both a **pattern and library** that helps developers implement **complex state management** requirements at scale.

Redux is the **most known state library** in the JavaScript community.

Basic concept

Store

Redux relies on **a single immutable object** to store all the application state.
This object is called a store. (You can compare it to a client-side database).

How?

State circuit

When a button is clicked an **action** will be **dispatched** which has **name** like an event and a **payload** with the data that it wants to change.
But the **store is immutable**...
So, to change the state of the application, an **entire new object is created**, by passing the **current state** and the **action payload** into a **reducer function**, which returns a new object, with the entire application state.

Steps

1. User **clicks** on a button.
2. Click event is **dispatched** as a **name**, along with the **data** as payload.
3. Since the **store is immutable**, we **create a new object** based on the **current state** (before user clicked the button) and the **action payload**.
4. The function that handles the **passing and creation** of the **new state** to the **store** is called a **reducer function**.

State in use

Create a new react project

$> npm create vite _

Install reduxjs toolkit

$> npm install reduxjs/toolkit react-redux _

Create a store

1. Create a **store** file inside your **src folder**.

2. Set up the **global store** object by using the **configureStore** function.
   It will **register any reducer** to find in the code.

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

export const store = configureStore({
    reducer: {
        pizza: pizzaReducer;
    }
});
```

Provide the store

Provider will make its **accessible** for the **entire component tree**.

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

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>
)
```

Create a pizza slice

Create a **pizza slice** to represent some **data** in the store.
It should have a **unique name**, an **initial state,** and a **collection of reducers**.
A reducer is a **function** that takes the **old state** and an **action**, then **define the logic** that's required to **change the state**.

```
const initialState = {
    toppings: ['cheese'],
    smokingHot: true
}

export const pizzaSlice = createSlice({
    name: 'pizza',
    initialState,
    reducers: {
        toggleHeat: (state) => {
            state.smokingHot = !state.smokingHot
        },
        addTopping: (state, action) => {
            state.toppings = [...state.toppings, action.payload];
        }
    }
})
```

Export defined reducer functions

Export the defined reducer functions so we can put them to use in our UI component.

```
export default pizzaSlice.reducer
```

Export correlated actions

Redux toolkit will automatically **generate actions** that corresponds the names of the **defined reducer functions**.

```
export const { toggleHeat, addTopping } = pizzaSlice.actions;
```

Display them pizzas!

With redux, we can now **select data** from **anywhere** inside the application without the need of context or prop drilling (see my react article).
We can now grab any **reactive value** or **slice** in the store with the **useSelector** hook.
To **change the application data**, an action needs to be **dispatched** to the store.

```
function App() {

  const pizza = useSelector(state => state.pizza);
  const dispatch = useDispatch();

  return (
    <>
      <h1>Pizza</h1>
      {
        pizza.toppings.map((topping, i) => (
          <div key={topping + i}>{topping}</div>
        ))
      }
      <button onClick={() => dispatch(addTopping('horse meat'))}>
        Add Horse meat
      </button>
    </>
  )
}

export default App
```

Run it!

$> npm run dev _

Extra

You can install the redux devtools browser extension to take a look at the entire timeline of actions and state changes in you application.

Made by :

Matthias Brat

Socials:

github.com/matthiasbrat

stackoverflow.com/users/17921879/matthias-b

matthiasbrat.dev