# Zustand vs Redux

# Simplicity

## Zustand

Offers minimal boilerplate and is very easy to set up, making it ideal for quick and straightforward state management.

```
import create from 'zustand';

const useStore = create((set) => ({
  count: 0,
  increase: () => set((state) => ({ count: state.count + 1 })),
  decrease: () => set((state) => ({ count: state.count - 1 })),
}));

function Counter() {
  const { count, increase, decrease } = useStore();
  return (
    <div>
      <button onClick={decrease}>-</button>
      <span>{count}</span>
      <button onClick={increase}>+</button>
    </div>
  );
}
```

Define store

Usage in component

# Simplicity

## Redux

Requires more boilerplate code, which can make initial setup more complex but offers a more structured approach to state management.

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

const INCREASE = 'INCREASE';
const DECREASE = 'DECREASE';

const increase = () => ({ type: INCREASE });
const decrease = () => ({ type: DECREASE });
```

Define action types

Define actions

```jsx
const counterReducer = (state = { count: 0 }, action) => {
  switch (action.type) {
    case INCREASE:
      return { ...state, count: state.count + 1 };
    case DECREASE:
      return { ...state, count: state.count - 1 };
    default:
      return state;
  }
};

const store = createStore(counterReducer);

function Counter() {
  const dispatch = useDispatch();
  const count = useSelector((state) => state.count);
  return (
    <div>
      <button onClick={() => dispatch(decrease())}>-</button>
      <span>{count}</span>
      <button onClick={() => dispatch(increase())}>+</button>
    </div>
  );
}

function App() {
  return (
    <Provider store={store}>
      <Counter />
    </Provider>
  );
}
```

Define reducer

Create store

Usage in component

Provide store to the app

# API

## Zustand

Provides an intuitive and minimalistic API, allowing developers to grasp its concepts and start using it with minimal effort.

## Redux

Has an extensive API with a steeper learning curve, but it provides powerful tools and patterns for managing state in larger applications.

Swipe ⟶

# Size

## Zustand

Is lightweight, leading to a smaller bundle size which can be beneficial for performance, especially in smaller applications.

## Redux

Due to its dependencies and additional libraries often used alongside it, Redux tends to result in a larger bundle size.

# Selectors

## Zustand

Built-in selectors in Zustand make it easy to manage and access specific parts of the state without needing additional libraries.

## Redux

Efficient selectors in Redux often require external libraries like Reselect, adding to the complexity but allowing for highly optimized state access.

# Middleware

## Zustand

While Zustand supports middleware, its ecosystem is less extensive compared to Redux, focusing more on simplicity.

## Redux

Has a very extensive middleware ecosystem, such as Redux Thunk and Redux Saga, which offers advanced capabilities for handling side effects and asynchronous operations.

Swipe ⟶


Subscribe on YouTube to learn more
Code with Sloba
@CodewithSloba • 21.8K subscribers
Hi, my name is Slobodan (Sloba) Gajic, and I'm a JavaScript software developer. Building a
Subscribed