10 USEFUL TIPS FOR MASTERING REDUX

AWAİS GENİUS

Swipe Right

KEY REDUX CONCEPTS ?

- Redux manages app state using a few key concepts. The **store** holds the entire state, while **actions** describe changes, and **reducers** handle those changes to update the state. To trigger updates, you use **dispatch** to send actions. **Middleware** can be added for tasks like handling async actions. **Selectors** help retrieve specific pieces of state, and the **Provider** component connects the **store** to your React app. Together, these concepts create a predictable and maintainable way to manage state in your application.

AWAİS GENİUS

Swipe Right

UNDERSTAND THE REDUX FLOW !

- **Actions:** Plain objects that describe what happened.
- **Reducers:** Pure functions that take the current state and an action, and return a new state
- **Store:** Holds the entire state of the app.

AWAİS GENİUS

Swipe Right

TİP #1 - UNDERSTAND ACTİONS, REDUCERS, AND STORE

```
// Action
const increment = { type: 'INCREMENT' };

// Reducer
const counter = (state = 0, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return state + 1;
    default:
      return state;
  }
};

// Store
import { createStore } from 'redux';
const store = createStore(counter);

store.dispatch(increment); // State: 1
```

AWAİS GENİUS

Swipe Right

TİP #2 - KEEP STATE IMMUTABLE

- **Return a New State, Don’t Mutate**
- **Store:** Holds the entire state of the app.

```
// Bad Example (Mutating State)
state.counter++;

// Good Example (Immutable)
return { ...state, counter: state.counter + 1 };
```

AWAİS GENİUS

Swipe Right

TİP #3 - SPLİT REDUCERS WİTH COMBİNEREDUCERS

- **Use createSlice to simplify reducer logic**
- **Example:**

```
import { createSlice } from "@reduxjs/toolkit";

const counterSlice = createSlice({
  name: "counter",
  initialState: { value: 0 },
  reducers: {
    increment: (state) => { state.value += 1; },
    decrement: (state) => { state.value -= 1; }
  }
});

export const { increment, decrement } = counterSlice.actions;
export default counterSlice.reducer;
```

AWAİS GENİUS

Swipe Right

TİP #4 - USE REDUX DEVTOOLS FOR DEBUGGİNG

Track Every Action

- Install Redux DevTools for real-time debugging.
- Inspect the state, dispatched actions, and time-travel debugging.

```
// Setting up Redux DevTools
const store = createStore(
  rootReducer,
  window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__()
);
```

AWAİS GENİUS

Swipe Right

TİP 5 - USE SELECTORS INSTEAD OF DİRECT STATE ACCESS 🎯

- Improves performance by memoizing derived state.
- Example with createSelector

```
import { createSelector } from '@reduxjs/toolkit';

const selectCounter = (state) => state.counter.value;

export const selectDoubleCounter = createSelector(
  selectCounter,
  (value) => value * 2
);
```

AWAİS GENİUS

Swipe Right

TİP 6 - USE MİDDLEWARE FOR ASYNC LOGİC ⏳

- Use redux-thunk or redux-saga for handling API calls
- Example with redux-thunk:

```
import { createAsyncThunk, createSlice } from "@reduxjs/toolkit";

export const fetchUsers = createAsyncThunk("users/fetch", async () => {
  const response = await fetch("https://api.example.com/users");
  return response.json();
});

const usersSlice = createSlice({
  name: "users",
  initialState: { list: [], status: "idle" },
  extraReducers: (builder) => {
    builder.addCase(fetchUsers.fulfilled, (state, action) => {
      state.list = action.payload;
    });
  }
});

export default usersSlice.reducer;
```

AWAİS GENİUS

Swipe Right

TİP 7 - KEEP ACTİONS DESCRİPTİVE

- Use meaningful action names to improve debugging.
- Bad ❌:

```
dispatch({ type: "UPDATE" });
```

- Good ✅:

```
dispatch({ type: "UPDATE_USER_PROFILE", payload: userData });
```

AWAİS GENİUS

Swipe Right

TİP 8 - NORMALİZE STATE FOR LARGE DATA

- Avoid deeply nested objects; use normalization for better performance.
- Example with normalizr:

```
import { schema, normalize } from "normalizr";

const user = new schema.Entity("users");
const myData = normalize(responseData, [user]);
```

AWAİS GENİUS

Swipe Right

TİP 9 - USE USEDİSPATCH AND USESELECTOR HOOKS 🔥

- Instead of connect(), use hooks for better performance.
- Example:

```
import { useSelector, useDispatch } from "react-redux";
import { increment } from "./counterSlice";

function Counter() {
  const count = useSelector((state) => state.counter.value);
  const dispatch = useDispatch();

  return (
    <div>
      <h1>{count}</h1>
      <button onClick={() => dispatch(increment())}>+</button>
    </div>
  );
}
```

AWAİS GENİUS

Swipe Right

TİP 10 - OPTİMİZE PERFORMANCE WİTH USEMEMO & USECALLBACK

- Use useMemo to avoid unnecessary re-sorting of users when rendering.
- Use useCallback to prevent unnecessary function recreation for event handlers.

```jsx
const UserList = () => {
  const users = useSelector((state) => state.users.list);
  const dispatch = useDispatch();
  const [name, setName] = useState("");

  const sortedUsers = useMemo(() =>
    [...users].sort((a, b) => a.name.localeCompare(b.name)), [users]
  );

  const handleAddUser = useCallback(() => {
    if (name.trim()) {
      dispatch(addUser({ id: Date.now(), name }));
      setName("");
    }
  }, [name, dispatch]);

  return (
    <div>
      <input value={name} onChange={(e) => setName(e.target.value)} />
      <button onClick={handleAddUser}>Add</button>
      <ul>{sortedUsers.map((user) => <li key={user.id}>{user.name}</li>)}</ul>
    </div>
  );
};
```

AWAİS GENİUS

Swipe Right

CONCLUSİON 🎉

- Redux is powerful but needs best practices to scale efficiently.
- Key takeaways:

✅ Use Redux Toolkit

✅ Structure store properly

✅ Optimize selectors & use middleware

💡 Ready to level up your Redux game? Drop your thoughts in the comments! 👇

AWAİS GENİUS

Thank you!