Redux Toolkit Workflow: Step-by-Step Guide

1. **Define the Feature Slice**

   Each **feature (state)** in your app should have its own **slice**. This slice:

   - Defines the initial state
   - Contains reducers (business logic)
   - Exports actions and reducer

Example: Creating a **counterSlice.js**

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

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

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

2. **Configure the Redux Store**

The **store** holds all slices in a structured way. Each **key** in the store represents a different **feature**.

Example: Setting up **store.js**

```
import { configureStore } from '@reduxjs/toolkit';
import counterReducer from './features/counterSlice';
import videoReducer from './features/videoSlice'; // Example: Another slice

const store = configureStore({
  reducer: {
    counter: counterReducer, // "counter" state
    video: videoReducer      // "video" state
  }
});

export default store;
```

3. **Provide the store to the React App**

To connect Redux to React, **wrap your app** with **Provider**, so all components can access the store.

Example: Adding **Provider** in **index.js**

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

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

4. Select State from the Redux Store using useSelector()

Components use the **useSelector()** hook to **access specific slices of state** from Redux.

Example: Accessing the Counter State

```
import { useSelector } from 'react-redux';

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

  return <h1>Counter: {count}</h1>;
};
```

5. Dispatch Actions using useDispatch()

Actions trigger state changes in Redux. Use the **useDispatch()** hook **to send actions to Redux**.

Example: Dispatching Actions to Change State

```
import { useDispatch } from 'react-redux';
import { increment, decrement } from '../features/counterSlice';

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

  return (
    <div>
      <button onClick={() => dispatch(increment())}>Increment</button>
      <button onClick={() => dispatch(decrement())}>Decrement</button>
    </div>
  );
};
```

6. Redux Forwards the Action to the Respective Reducer

- When *dispatch(increment())* is called, Redux **forwards** the action to the **counterSlice** reducer.

- The **reducer checks** the **action.type** and **updates the state accordingly**.

7. **The Reducer Updates the State**

   The reducer **modifies the state based on the action type**.

   Example: If *increment()* is dispatched

```
reducers: {
  increment: (state) => { state.count += 1; }
}
```

   - The **state updates** from **{ count: 0 }** -> **{ count: 1 }**.
   - This happens immutably using Immer.js (built into Redux Toolkit).

8. **Redux Updates the Store**

   Once the reducer modifies the state, Redux **updates the store** with the new state values.

9. **React Automatically Re-Renders Components**

   - Since, the store is updated, React detects the **state change** and **re-renders only the components using the modified state**.
   - This ensures performance without unnecessary re-renders.

10. **The Updated State Reflects the UI**

    After re-rendering, the latest state values are displayed in the UI.

    Example: Updated Counter Display

```
const CounterDisplay = () => {
  const count = useSelector((state) => state.counter.count);

  return <h1>Counter: {count}</h1>; // Updated count will be shown here
};
```

Final Summary of the Redux Toolkit Workflow:

- Create a Slice for each feature (state).
- Define Reducers (Business Logic) inside the slice.
- Export Actions & Reducers from the slice.
- Configure the Store by adding all feature slices.
- Provide the Store to the App by using the <Provider />.

- Use the useSelector() hook Read State inside the components.
- Use the useDispatch() hook to Send Actions to Redux.
- Reducer Modifies the State based on the action type.
- Redux updates the Store with new state values.
- React Detects the State Change and re-renders the components efficiently.

Why this Guide Works for Memorization

- Structured Steps: Clearly defined, numbered steps make it easy to recall.
- Code Examples: Each step has an example, making it practical.
- Logical Flow: Every step follows naturally from the previous one.
- Review-Friendly: Read this guide a few times, and it will become second nature.

Tip: Try rewriting these steps from memory after reviewing them 3 to 4 times. It will help reinforce the process in your brain.