React Redux Toolkit - RTK

In Detail

@AnuragSingh

What is Redux Toolkit ?

A **set of tools** to simplify working with Redux, a state management library for JavaScript apps.

Aims to **reduce boilerplate code**, make code more **readable**, and enforce best practices.

Easier state management in React components, **Recommended** as the standard way to write Redux logic.

@AnuragSingh

RTK Functions & Utilities

Most useful functions, components, and methods

- configureStore( )
- <Provider />

- useSelector( )
- useDispatch( )

- createSlice( )
- extraReducers

- createAsyncThunk( )

@AnuragSingh

1. configureStore( )

- **Creates the Redux store** with recommended defaults.
- Automatically includes **middleware** like Redux **Thunk** for **async** actions.

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

const store = configureStore({
  reducer: {
    /* your reducers */
  }
});

export default store;
```

Example

@AnuragSingh

2. <Provider>

@AnuragSingh

- **Provides access** to the Redux store throughout a React application.

- Acts **as a bridge** between React components and the Redux store.

- **Eliminates** the need for manual **prop drilling** to pass state down the component tree.

```jsx
import { Provider } from 'react-redux';
import store from './store'; // Your Redux store instance

function App() {
  return (
    <Provider store={store}>
      {/* Your app components here */}
    </Provider>
  );
}
```

Example

3. createSlice( )

@AnuragSingh

- Simplifies **reducer creation** for a specific slice of state. Generates **action creators** and **action types** automatically. Handles **immutable** updates with Immer

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

const counterSlice = createSlice({
  name: 'counter',
  initialState: { value: 0 },
  reducers: {
    increment: (state) ⇒ {
      state.value += 1;
    },
  },
});

// Export the action creator
export const { increment } = counterSlice.actions;

// Export the reducer
export default counterSlice.reducer;
```

Example

4. useDispatch

@AnuragSingh

- React hook for **dispatching actions** from components.

```
const dispatch = useDispatch();
dispatch(increment());
```

5. useSelector

- React hook for **accessing Redux state** in components.

```
const count = useSelector((state) ⇒
   state.counter.value
);
```

6. createAsyncThunk( )

- **createAsyncThunk** :
  - Part of Redux Toolkit.
  - Generates asynchronous action creators.
  - Takes an **action type** string and an **async function.**

- **Action Type** :
  - First argument to **createAsyncThunk**.
  - String format: "posts/fetchPosts".
  - Represents action type during async operation stages.

- **Async Function** :
  - Second argument to **createAsyncThunk**.
  - Contains asynchronous logic (e.g., fetching data).

@AnuragSingh

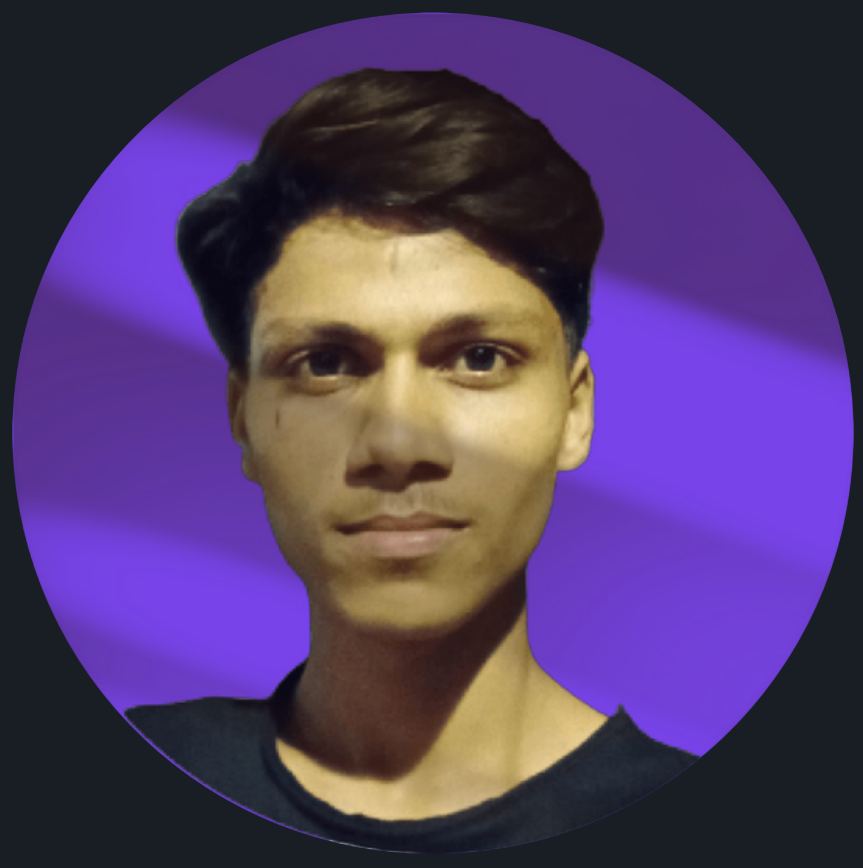
Example

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

// Async thunk function
export const fetchData = createAsyncThunk("data/fetchData", async () => {
  try {
    // Simulating an API request
    const response = await fetch("https://example.com/todos/1");
    const data = await response.json();
    return data;
  } catch (error) {
    // If there's an error during the API request
    throw new Error("Failed to fetch data");
  }
});

// Slice
const dataSlice = createSlice({
  name: "data",
  initialState: {
    status: "idle", // 'idle', 'loading', 'succeeded', 'failed'
    error: null,
    data: null
  },
  reducers: {},
  extraReducers: (builder) => {
    builder
      .addCase(fetchData.pending, (state) => {
        state.status = "loading";
      })
      .addCase(fetchData.fulfilled, (state, action) => {
        state.status = "succeeded";
        state.data = action.payload;
      })
      .addCase(fetchData.rejected, (state, action) => {
        state.status = "failed";
        state.error = action.error.message;
      });
  }
});

// Export the async thunk and the reducer
export { fetchData };
export default dataSlice.reducer;
```



@AnuragSingh

FOLLOW