Redux ToolKit (RTK) Guide

Evolution:

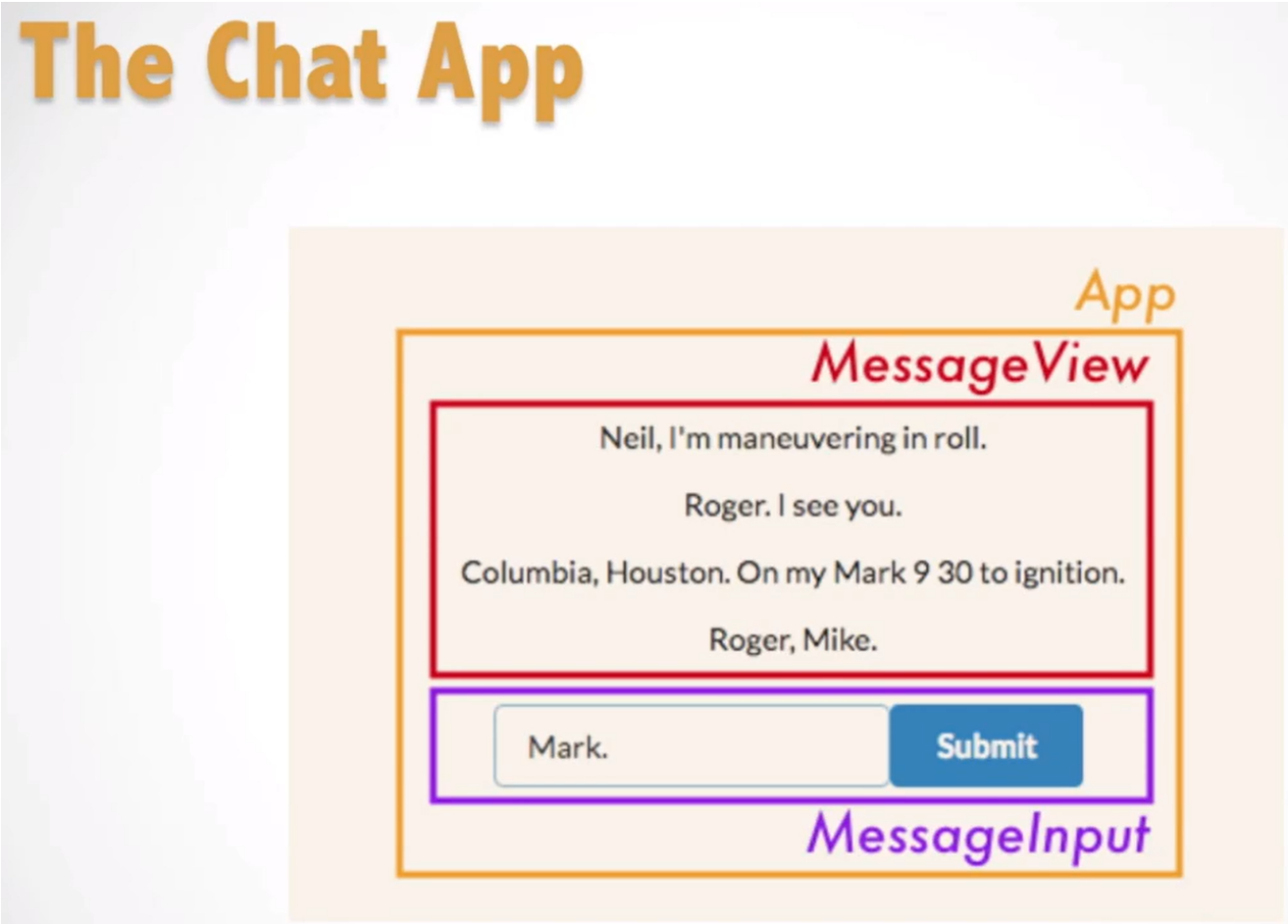
1. Redux - A predictable state container for JavaScript applications
2. React-Redux - Official React bindings for Redux (includes the hooks useDispatch and useSelector)
3. Redux ToolKit (RTK) - The official, opinionated, batteries-included toolset for efficient Redux development

---

Instalation:

```
npm install @reduxjs/toolkit react-redux
```

'RTK Chat App'



Application files/folder structure (with ViteJS)

- `/src`
  - `main.jsx`: the starting point for the app
  - `App.jsx`: the top-level React component
  - `/app`: includes the app dependencies
    - `store.js`: creates the Redux store instance (manages the global state of the app)
  - `/features`: organizes the features of the app
    - `/chat`: the "chat" feature folder
      - `/components`: includes React components of the "chat" feature
        - `Chat.jsx`: main React component for the "chat" feature
        - `MessageInput.jsx`: partial React component for the "chat" feature
        - `MessageView.jsx`: partial React component for the "chat" feature
      - `chatSlice.js`: the Redux logic for the "chat" feature (used to define the reducers object with functions for the actions [add, delete, clear])

Application Implementation Steps

1. Creating the Redux Store

   src/app/store.js

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

   export default configureStore({
     reducer: {
       // to fill with one of each feature slice reducer
     }
   });
   ```

   The Redux store is created using the `configureStore` function from Redux Toolkit. `configureStore` requires that we pass in a `reducer` object as argument.

2. Import the `Chat` feature component and include it in the `App` top-level component

   src/App.jsx

```
import Chat from './features/chat/components/Chat';
import './App.css';

export default function App() {
    return (
        <div className='App'>
            <h1>RTK Chat App</h1>
            <Chat />
        </div>
    );
}
```

3. Make the Redux store available to the entire app

   src/main.jsx

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

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

   - Import `Provider` from the `react-redux` library
   - Surround the `<App />` component with the `<Provider><Provider/>` tags
   - In the `<Provider><Provider/>` tag we must include a `store={}` prop pointing to the `src/app/store.js` file (make sure it's imported on top of the `main.jsx` file)

4. `Chat` feature components starting code:

   src/features/chat/components/Chat.jsx

```
import MessageView from './MessageView';
import MessageInput from './MessageInput';
```

```
export default function Chat() {
    return (
        <section className='chat'>
            <h1>Chat</h1>
            <button>Clear All</button>

            <article className='message-view'>
                <MessageView />
            </article>

            <article className='message-input'>
                <MessageInput />
            </article>
        </section>
    );
}
```

src/features/chat/components/MessageView.jsx

```
export default function MessageView() {
    return (
        <section>
            <h1>Message View</h1>
            <p>msg 1</p>
            <p>msg 2</p>
        </section>
    );
}
```

src/features/chat/components/MessageInput.jsx

```
export default function MessageInput() {
    return (
        <section>
            <h1>Message Input</h1>
            <input type='text' placeholder='Enter your message' />
            <button>Send</button>
        </section>
    );
}
```

5. Creating a slice to the `Chat` feature

src/features/chat/components/chatSlice.js

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

const chatSlice = createSlice({
  // Identifier for this slice (name)
  name: "messages",
  initialState: {
    messages: [],
  },
  // The reducers object is used to define the actions (methods) that wil
  // These actions will then be exposed in the reducer "store" object, th
  reducers: {
    // Each action/method takes as arguments the current "state" of the "
    addMessage: (state, action) => {
      // Redux Toolkit allows us to write "mutating" logic in reducers (p
      // because it uses the "immer" library, which detects changes to a
      // immutable state based off those changes

      // The "action" has a "payload" property that will contain the valu
      state.messages.push(action.payload);
    },
    deleteMessage: (state, action) => {
      state.messages = state.messages.filter((msg, idx) => idx !== action
    },
    clearAllMessages: (state) => {
      state.messages = [];
    }
  }
});

// Export the actions of this "slice", destructuring them via the "action
export const { addMessage, deleteMessage, clearAllMessages } = chatSlice.
// Export the "reducer" from this "slice"
export default chatSlice.reducer;
```

A "slice" is a collection of Redux reducer logic and actions for a single feature in your app, typically defined together in a single file. The name comes from splitting up the root Redux state object into multiple "slices" of state.

6. Connect the `chatSlice` reducer with the `store` reducer

   src/app/store.js

```
import { configureStore } from '@reduxjs/toolkit';
import chatSlice from '../features/chat/chatSlice';

export default configureStore({
  reducer: {
    // one connection for each feature's slice reducer
    chat: chatSlice,
  }
});
```

- Make sure `chatSlice` is imported above.

7. Redux usage through the hooks `useDispatch( )` (to update the state) and `useSelector( )` (to read the state) from the `react-redux` library

   src/features/chat/components/MessageInput.jsx

```
import { useState } from 'react';
import { useDispatch } from 'react-redux';
import { addMessage } from '../chatSlice';

export default function MessageInput() {
    const [msgTxt, setMsgTxt] = useState('');
    // Create a dispatch method to use the "actions" and send them to the
    let dispatch = useDispatch();

    const handleSendMsg = () => {
        // Use the dispatch method to call the action/method "addMessage(
        dispatch(addMessage(msgTxt));
        setMsgTxt('');
    };

    return (
        <section>
            <h1>Message Input</h1>
            <input
                type='text'
                placeholder='Enter your message'
                value={msgTxt}
                onChange={({ target: { value } }) => setMsgTxt(value)}
            />
            <button onClick={handleSendMsg}>Send</button>
        </section>


    );
```

```
}
```

src/features/chat/components/MessageView.jsx

```
import { useState } from 'react';
import { useDispatch } from 'react-redux';
import { addMessage } from '../chatSlice';

export default function MessageInput() {
    const [msgTxt, setMsgTxt] = useState('');
    // Create a dispatch method to use the "actions" and send them to the
    let dispatch = useDispatch();

    const handleSubmit = (e) => {
        e.preventDefault();
    };

    const handleSendMsg = () => {
        // Use the dispatch method to call the action/method "addMessage(
        dispatch(addMessage(msgTxt));
        setMsgTxt('');
    };

    return (
        <section>
            <h1>Message Input</h1>
            <form onSubmit={handleSubmit}>
                <input
                    type='text'
                    placeholder='Enter your message'
                    value={msgTxt}
                    onChange={({ target: { value } }) => setMsgTxt(value)
                />
                <button onClick={handleSendMsg}>Send</button>
            </form>
        </section>
    );
}
```

src/features/chat/components/Chat.jsx

```
import MessageView from './MessageView';
import MessageInput from './MessageInput';
import { useDispatch } from 'react-redux';
```

```
import { clearAllMessages } from '../chatSlice';

export default function Chat() {
    let dispatch = useDispatch();

    const handleClearAll = () => {
        dispatch(clearAllMessages());
    };

    return (
        <section className='chat'>
            <h1>Chat</h1>
            <button onClick={handleClearAll}>Clear All</button>

            <article className='message-view'>
                <MessageView />
            </article>

            <article className='message-input'>
                <MessageInput />
            </article>
        </section>
    );
}
```