Tổng Hợp Kiến Thức

1. Khi Nào Cần Redux Toolkit?

Trạng Thái Cục Bộ (Local State)

- **Dùng** `useState`**:** Quản lý trong component.
- **Phạm vi:** Chỉ component đó và con trực tiếp (qua props).
- **Ví dụ:** Giá trị input, trạng thái toggle menu.

Trạng Thái Toàn Cục (Global State)

- **Dùng Redux Toolkit:** Quản lý ngoài cây component.
- **Khi nào cần:**
  - Chia sẻ trạng thái giữa nhiều component ở các nhánh khác nhau.
  - Ứng dụng lớn, phức tạp.
  - Dữ liệu cần duy trì xuyên suốt vòng đời.
  - Tổ chức logic rõ ràng, dễ mở rộng.
- **Ví dụ:** Danh sách công việc (todos), trạng thái bộ lọc.

Redux Toolkit vs Context API

Tiêu chí	Context API	Redux Toolkit
Độ phức tạp	Đơn giản	Ban đầu phức tạp, RTK đơn giản hóa
Mở rộng	Nhỏ/vừa	Lớn/phức tạp
Hiệu năng	Có thể kém (re-render nhiều)	Tốt (selectors tối ưu)
Debug	Khó	Dễ (Redux DevTools)
Trường hợp	Theme, ngôn ngữ	Trạng thái phức tạp, nghiệp vụ

---

2. Giới Thiệu Redux Toolkit

Redux Toolkit Là Gì?

- Thư viện cải tiến Redux, giảm phức tạp, tích hợp best practices.
- Dùng trong React để quản lý trạng thái toàn cục hiệu quả.

Thành Phần Chính

- **Store:** Kho lưu trữ trạng thái toàn cục.
- **Slice:** Quản lý trạng thái/logic cho một feature (VD: `todoSlice`).
  - **Initial State:** Trạng thái ban đầu.
  - **Reducers:** Logic cập nhật trạng thái.
  - **Actions:** Sự kiện gửi đến reducer.

Luồng Dữ Liệu

Component → Dispatch Action → Slice Reducer → Store → Component (re-re
nder

---

3. Xây Dựng Store và Slice

Tạo Slice với `createSlice`

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

const todoSlice = createSlice({
  name: 'todos',
  initialState: {
    todos: [],
  },
  reducers: {
    addTodo: (state, action) => {
      state.todos.push(action.payload); // Mutate trực tiếp, ImmerJS đảm bảo im
mutability
    },
```

```
    toggleTodo: (state, action) => {
      const todo = state.todos.find(t => t.id === action.payload);
      if (todo) todo.completed = !todo.completed;
    },
  },
});

export const { addTodo, toggleTodo } = todoSlice.actions;
export default todoSlice.reducer;
```

Thiết Lập Store với `configureStore`

```
import { configureStore } from '@reduxjs/toolkit';
import todoReducer from './slices/todoSlice';

const store = configureStore({
  reducer: {
    todos: todoReducer,
  },
});

export default store;
```

---

4. Kết Nối Component với Store

Cung Cấp Store với `Provider`

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

ReactDOM.createRoot(document.getElementById('root')).render(
```

```
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>
);
```

Truy Cập State với `useSelector`

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

function TodoList() {
  const todos = useSelector(state => state.todos.todos);

  return (
    <ul>
      {todos.map(todo => (
        <li key={todo.id}>{todo.text}</li>
      ))}
    </ul>
  );
}
```

---

5. Cập Nhật Trạng Thái với `dispatch`

Dispatch Action với `useDispatch`

```
import { useState } from 'react';
import { useDispatch } from 'react-redux';
import { addTodo } from '../redux/slices/todoSlice';

function TodoInput() {
  const dispatch = useDispatch();
  const [text, setText] = useState('');
```

```
  const handleAdd = () => {
    if (text.trim()) {
      dispatch(addTodo({ id: Date.now(), text, completed: false }));
      setText('');
    }
  };

  return (
    <div>
      <input value={text} onChange={e => setText(e.target.value)} />
      <button onClick={handleAdd}>Add Todo</button>
    </div>
  );
}
```

Action Creators

- **Được tạo tự động bởi `createSlice`:**
  - `addTodo(todo)`: Gửi payload là object todo mới.
  - `toggleTodo(id)`: Gửi payload là ID của todo cần toggle.

Luồng Cập Nhật

Component → dispatch(action) → Reducer → Store → Re-render (useSelecto
r)

Tham Chiếu và Re-render

- Redux Toolkit (ImmerJS) tạo tham chiếu mới khi trạng thái thay đổi.
- `useSelector` so sánh tham chiếu, kích hoạt re-render khi thay đổi.