Chapter 11: Quản lý State toàn cục với Redux Toolkit

A. Mục tiêu

Sau bài học này, học viên sẽ:

- Hiểu khi nào cần sử dụng **Redux Toolkit (RTK)** để quản lý state toàn cục trong ứng dụng React/Next.js.
- Nắm vững các khái niệm cốt lõi của RTK: **configureStore**, **createSlice**, **store**, **dispatch**, và **selector**.
- Biết cách tạo một **slice** để quản lý một phần của state toàn cục.
- Tích hợp Redux store vào ứng dụng Next.js và sử dụng các hook như **useSelector** và **useDispatch** trong component.
- Hiểu cách sử dụng **Redux DevTools** để debug state và actions.
- Áp dụng RTK vào các ứng dụng thực tế một cách hiệu quả, giảm thiểu boilerplate code.

B. Nội dung lý thuyết

1. Tại sao cần Redux?

Trong các ứng dụng React, **useState** và **useContext** (kết hợp với **useReducer**) thường đủ để quản lý state cho các ứng dụng vừa và nhỏ. Tuy nhiên, khi ứng dụng trở nên phức tạp, Redux trở thành lựa chọn tối ưu trong các trường hợp sau:

- **State toàn cục lớn và phức tạp**: Nhiều component cần truy cập và cập nhật cùng một state.
- **Logic bất đồng bộ phức tạp**: Cần xử lý các API call, side effects, hoặc middleware.
- **Debugging mạnh mẽ**: Redux DevTools cho phép theo dõi lịch sử state và actions.

- **Dự án lớn với nhiều lập trình viên**: Redux cung cấp cấu trúc rõ ràng để quản lý state, dễ dàng bảo trì và mở rộng.

Hạn chế của useContext + useReducer:

- **Khó mở rộng**: Khi state lớn, logic trong reducer trở nên phức tạp.
- **Hiệu năng**: useContext có thể gây re-render không cần thiết.
- **Thiếu công cụ debug**: Không có công cụ như Redux DevTools để theo dõi thay đổi state.

Ưu điểm của Redux:

- State được quản lý tập trung trong một **store** duy nhất.
- Dễ dàng debug với Redux DevTools.
- Hỗ trợ middleware (như Redux Thunk) để xử lý logic bất đồng bộ.
- Redux Toolkit giảm thiểu boilerplate code so với Redux gốc.

---

2. Redux Toolkit (RTK) - Redux thời hiện đại

Redux Toolkit (RTK) là bộ công cụ chính thức của Redux, được thiết kế để đơn giản hóa việc viết code Redux, giảm boilerplate và cải thiện trải nghiệm lập trình.

Các khái niệm cốt lõi:

- **configureStore**: Hàm tạo store, tự động tích hợp Redux DevTools và middleware (như Redux Thunk).

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

const store = configureStore({
  reducer: {
    // Các reducer sẽ được thêm vào đây
```

```
  },
});
```

- **createSlice**: Hàm chính để tạo một "slice" (một phần của state). Nó tự động tạo reducer và action creators.

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

const counterSlice = createSlice({
  name: 'counter',
  initialState: { value: 0 },
  reducers: {
    increment(state) {
      state.value += 1; // Mutate trực tiếp nhờ Immer
    },
    decrement(state) {
      state.value -= 1;
    },
    incrementByAmount(state, action) {
      state.value += action.payload;
    },
  },
});
```

- **Store**: Object duy nhất chứa toàn bộ state của ứng dụng.
- **Dispatch**: Hàm để gửi action đến store, kích hoạt reducer để cập nhật state.
- **Selector**: Hàm để trích xuất dữ liệu từ store.

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

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

- **Immer**: Thư viện tích hợp trong RTK, cho phép viết code "mutate" state một cách an toàn (thực chất tạo bản sao mới của state).

Ưu điểm của RTK:

- Giảm boilerplate so với Redux gốc (không cần viết action types hay action creators thủ công).
- Tích hợp sẵn Redux DevTools và middleware.
- Code dễ đọc, dễ bảo trì nhờ `createSlice` và Immer.

---

3. Tích hợp Redux với React/Next.js

Để sử dụng Redux trong ứng dụng Next.js, cần thực hiện các bước sau:

1. **Cài đặt store** và cung cấp cho ứng dụng thông qua `<Provider>` (từ `react-redux`).
2. Sử dụng **useSelector** để đọc state từ store.
3. Sử dụng **useDispatch** để gửi action đến store.

Ví dụ tích hợp:

- Bao bọc ứng dụng bằng `<Provider>` trong `pages/_app.js`:

```
import { Provider } from 'react-redux';
import store from '../store';

export default function MyApp({ Component, pageProps }) {
  return (
    <Provider store={store}>
      <Component {...pageProps} />
    </Provider>
  );
}
```

- Sử dụng trong component:

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

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

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => dispatch(increment())}>Tăng</button>
      <button onClick={() => dispatch(decrement())}>Giảm</button>
    </div>
  );
}
```

Lưu ý khi dùng Redux trong Next.js:

- Redux hoạt động tốt với cả **Server-Side Rendering (SSR)** và **Static Site Generation (SSG)**, nhưng cần cấu hình đúng để tránh hydrate lỗi.
- Đảm bảo store được tạo một lần duy nhất (trong file `store.js`) và chia sẻ cho cả client và server.

C. Bài tập thực hành

Bài 1: Cài đặt Redux Toolkit và react-redux

1. Cài đặt các package:

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

Bài 2: Cấu hình store

1. Tạo file `store.js` trong thư mục gốc:

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

export const store = configureStore({
  reducer: {},
  devTools: process.env.NODE_ENV !== 'production', // Bật DevTools ở môi
});
```

2. Kiểm tra cài đặt bằng cách import `store` trong `pages/_app.js`.

Bài 3: Tạo slice cho Counter

1. Tạo file `features/counter/counterSlice.js`:

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

const counterSlice = createSlice({
  name: 'counter',
  initialState: {
    value: 0,
  },
  reducers: {
    increment(state) {
      state.value += 1;
    },
    decrement(state) {
      state.value -= 1;
    },
    incrementByAmount(state, action) {
      state.value += action.payload;
    },
  },
});
```

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

2. Kết nối reducer vào store trong `store.js`:

```
import { configureStore } from '@reduxjs/toolkit';
import counterReducer from './features/counter/counterSlice';

export const store = configureStore({
  reducer: {
    counter: counterReducer,
  },
  devTools: process.env.NODE_ENV !== 'production',
});
```

Bài 4: Kết nối store với ứng dụng

1. Cập nhật `pages/_app.js`:

```
import { Provider } from 'react-redux';
import { store } from '../store';

export default function MyApp({ Component, pageProps }) {
  return (
    <Provider store={store}>
      <Component {...pageProps} />
    </Provider>
  );
}
```

2. Kiểm tra: Mở DevTools → Redux tab để xác nhận store được khởi tạo.

Bài 5: Sử dụng Redux trong component

1. Tạo file `components/Counter.js` :

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

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

  return (
    <div className="max-w-md mx-auto mt-10">
      <h2 className="text-2xl font-bold mb-4">Counter</h2>
      <p className="text-lg mb-4">Giá trị: {count}</p>
      <div className="space-x-2">
        <button
          onClick={() => dispatch(increment())}
          className="bg-blue-500 text-white px-4 py-2 rounded hover:bg-b
        >
          Tăng
        </button>
        <button
          onClick={() => dispatch(decrement())}
          className="bg-red-500 text-white px-4 py-2 rounded hover:bg-re
        >
          Giảm
        </button>
        <button
          onClick={() => dispatch(incrementByAmount(5))}
          className="bg-green-500 text-white px-4 py-2 rounded hover:bg-
        >
          Tăng 5
        </button>
      </div>
    </div>
  );
}
```

2. Sử dụng component trong `pages/index.js` :

```jsx
import Counter from '../components/Counter';

export default function Home() {
  return (
    <div className="min-h-screen flex items-center justify-center">
      <Counter />
    </div>
  );
}
```

3. Kiểm tra: Mở trình duyệt, nhấn các nút và quan sát giá trị thay đổi. Dùng Redux DevTools để xem lịch sử actions.

D. Bổ sung: Tối ưu hóa và Debug với Redux DevTools

1. Sử dụng Redux DevTools

- Cài đặt extension Redux DevTools trên Chrome hoặc Firefox.
- Mở DevTools → Tab Redux để:
  - Xem toàn bộ state của store.
  - Theo dõi lịch sử actions và payload.
  - "Time travel" để quay lại trạng thái trước đó.
- Ví dụ: Khi nhấn nút "Tăng", bạn sẽ thấy action `counter/increment` được dispatch.

2. Tối ưu hóa hiệu năng

- **Memoized selectors**: Sử dụng `createSelector` từ RTK để tối ưu hóa selector:

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

```
const selectCounter = (state) => state.counter;
export const selectCount = createSelector([selectCounter], (counter) =>
```

- **Bất đồng bộ với Redux Thunk**: RTK tích hợp sẵn Redux Thunk để xử lý API calls:

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

export const fetchData = createAsyncThunk('counter/fetchData', async ()
  const response = await fetch('https://api.example.com/data');
  return await response.json();
});
```

3. Xử lý SSR trong Next.js

- Khi dùng Redux với SSR, cần đảm bảo store được khởi tạo đúng cách để tránh hydrate lỗi:

```
// pages/_app.js
import { Provider } from 'react-redux';
import { store } from '../store';

export default function MyApp({ Component, pageProps }) {
  return (
    <Provider store={store}>
      <Component {...pageProps} />
    </Provider>
  );
}
```

E. So sánh useContext + useReducer và Redux Toolkit

Tiêu chí	useContext + useReducer	Redux Toolkit
Độ phức tạp	Đơn giản, phù hợp ứng dụng nhỏ	Phức tạp hơn, phù hợp ứng dụng lớn
Hiệu năng	Có thể gây re-render không cần thiết	Tối ưu hóa với Immer, ít re-render
Debugging	Không có công cụ debug chuyên dụng	Redux DevTools mạnh mẽ
Bất đồng bộ	Cần tự viết logic async	Tích hợp Redux Thunk, hỗ trợ createAsyncThunk
Boilerplate	Ít code, nhưng khó mở rộng khi state lớn	Ít boilerplate nhờ RTK, dễ mở rộng

---

F. Tài liệu tham khảo

- Redux Toolkit Documentation
- React-Redux Documentation
- Next.js with Redux
- Redux DevTools

G. Bài tập nâng cao (tùy chọn)

1. **Tích hợp API bất đồng bộ**:
    - Tạo một slice mới để quản lý state của dữ liệu API (ví dụ: danh sách bài viết).
    - Sử dụng `createAsyncThunk` để gọi API từ `jsonplaceholder.typicode.com`.

- Hiển thị danh sách bài viết trong component và xử lý trạng thái loading/lỗi.

2. **Tùy chỉnh selector**:
   - Sử dụng `createSelector` để tạo selector tính toán giá trị từ state (ví dụ: `isCounterPositive`).
   - Hiển thị thông báo khi giá trị counter lớn hơn 0.
3. **Form với Redux**:
   - Tạo một form sử dụng Redux để quản lý state (thay vì `useState` hoặc `react-hook-form`).
   - Dùng RTK để lưu trữ dữ liệu form và xử lý submit.
4. **Debug với Redux DevTools**:
   - Viết một báo cáo ngắn (200 từ) về cách sử dụng Redux DevTools để debug ứng dụng. Bao gồm cách xem lịch sử actions và cách "time travel".