Redux và Redux-Saga là hai công cụ quản lý trạng thái và xử lý side effects.

**1. Redux**

Redux là một thư viện quản lý trạng thái trong các ứng dụng JavaScript. Mục tiêu của Redux là lưu trữ và quản lý trạng thái toàn cục của ứng dụng một cách có tổ chức, dễ kiểm soát và dễ debug.

**Redux phù hợp khi:**

* Ứng dụng có nhiều component cần chia sẻ dữ liệu chung.
* Cần quản lý trạng thái phức tạp và dễ kiểm soát luồng dữ liệu.
* Muốn debug dễ hơn bằng cách lưu lịch sử các action và state.

**Các thành phần chính của Redux:**

* Store

Là nơi lưu trữ toàn bộ trạng thái (state) của ứng dụng.

* Reducer

Là hàm thuần nhận vào state hiện tại và một action, trả về state mới.

const counterReducer = (state = { count: 0 }, action) => {

  switch (action.type) {

    case 'INCREMENT':

      return { count: state.count + 1 };

    default:

      return state;

  }

};

* Action

Là một object mô tả hành động muốn thực hiện (phải có thuộc tính type).

{ type: 'INCREMENT' }

* Dispatch

Là hàm để gửi action đến reducer và cập nhật lại state trong store.

dispatch({ type: 'INCREMENT' });

**2. Redux-Saga**

Redux Saga là một middleware cho Redux giúp xử lý các tác vụ bất đồng bộ (async) như gọi API, delay, thao tác phức tạp,… một cách có tổ chức, dễ test và dễ kiểm soát luồng dữ liệu.

Redux Saga cho phép viết side effect logic (như gọi API, delay, lắng nghe hành động khác…) bằng cách sử dụng generator function (function\*), với cú pháp giống như async/await nhưng linh hoạt và mạnh hơn.

Quản lý luồng logic phức tạp: Dễ mô hình hóa các quy trình như: đăng nhập → load data → chuyển trang

Theo dõi action liên tục: takeEvery, takeLatest, takeLeading, v.v.

Dễ test: Vì saga là generator function, dễ kiểm tra từng bước

Dừng, hủy giữa chừng: Có thể cancel một saga đang chạy

**Generator function**

function\* mySaga() {

  yield 1;

  yield 2;

}

**Effects của Redux Saga** (giống async/await nhưng dùng yield)

call(fn, ...args): Gọi hàm async

put(action): Dispatch action

take(actionType): Chờ một action xảy ra

takeEvery: Lắng nghe và xử lý tất cả action

takeLatest: Chỉ giữ lại lần gọi gần nhất

select(selector): Lấy state từ store

delay(ms) Delay thời gian

**Một số helper quan trọng**

takeEvery: Mỗi action đều tạo một saga mới

takeLatest: Nếu có nhiều action cùng loại → hủy cũ, chạy mới

takeLeading: Chạy đầu tiên, bỏ qua những cái sau cho tới khi xong

fork, spawn: Chạy saga song song, không chờ

cancel, cancelled: Hủy giữa chừng saga