**1.Redux là gì? Tại sao phải sử dụng Redux.**

- **Redux** là một **thư viện quản lý trạng thái (state management)** dành cho các ứng dụng JavaScript, đặc biệt phổ biến khi dùng với **React**. Redux cho phép lưu trữ toàn bộ **state toàn cục (global state)** của ứng dụng trong một nơi duy nhất gọi là **Store**, giúp quản lý và chia sẻ dữ liệu giữa các component một cách hiệu quả và rõ ràng.

**- Redux rất hữu ích trong những trường hợp sau:**

**Khi nào cần dùng Redux?**

* Ứng dụng có nhiều component cần dùng chung dữ liệu.
* Tránh truyền props qua nhiều cấp (props drilling), giúp code sạch và dễ bảo trì.
* Cần quản lý trạng thái rõ ràng, dễ kiểm soát, dễ debug (sử dụng Redux DevTools).
* Cần xử lý các tác vụ bất đồng bộ như gọi API, loading, xử lý lỗi.

**Lợi ích khi sử dụng Redux:**

* Tạo ra luồng dữ liệu một chiều (predictable state) – dễ hiểu, dễ theo dõi.
* Giúp debug và kiểm thử dễ dàng hơn.
* Tăng khả năng tái sử dụng và mở rộng ứng dụng.
* Có thể tích hợp middleware để xử lý logic phức tạp (API, log, xác thực...).

**2. Các thành phần chính của redux? Flow hoạt động của redux?**

**- Có 4 thành phần cốt lõi:**

+ Actions

+ Reducers

+ Store

+ Middleware

**- Luồng hoạt động trong Redux diễn ra theo một chiều (One-Way Data Flow):**

1. Component tạo và gửi action bằng dispatch(action).
2. Action được gửi đến Reducer.
3. Reducer xử lý action và trả về state mới.
4. Store cập nhật state mới.
5. Các component đang dùng state sẽ được cập nhật lại giao diện (re-render nếu cần).

**- Sơ đồ tổng quát:**

Component → dispatch(action) → Reducer → Store (state mới) → Component re-render

## **3. Redux Actions**

### ➤ Khái niệm:

* **Action** là một đối tượng JavaScript thuần (plain object) dùng để mô tả một hành động xảy ra trong ứng dụng (ví dụ: thêm dữ liệu, xóa dữ liệu...).
* Mỗi action **bắt buộc phải có** thuộc tính type – thể hiện loại hành động. Ngoài ra có thể kèm theo dữ liệu phụ (payload).

### ➤ Mục đích:

* Là **tín hiệu** để yêu cầu Redux cập nhật state.
* Truyền thông tin từ **component** đến **Redux Store**.

## **Redux Reducer**

### ➤ Khái niệm:

* **Reducer** là một hàm thuần (**pure function**) nhận vào 2 đối số:
  1. state hiện tại.
  2. action được gửi tới.
* Dựa vào type của action, reducer **xử lý logic** và trả về **state mới**.

### ➤ Đặc điểm:

* Không thay đổi trực tiếp state cũ → phải trả về **bản sao mới** (tuân thủ nguyên tắc **immutable**).
* Không thực hiện side-effect (như gọi API, log, v.v.).

### ➤ Mục đích:

* Là nơi xử lý và xác định **state sẽ thay đổi như thế nào** dựa vào action.

## **Redux Store**

### ➤ Khái niệm:

* **Store** là một **kho lưu trữ trung tâm** toàn bộ state của ứng dụng.
* Chỉ có **một store duy nhất** trong một ứng dụng Redux.

### ➤ Mục đích:

* **Lưu trữ state toàn cục**.
* Quản lý việc cập nhật và truy xuất state.

### ➤ Các chức năng chính:

* getState() – Lấy state hiện tại.
* dispatch(action) – Gửi action để thay đổi state.
* subscribe(listener) – Theo dõi thay đổi của state.

## **Redux Middleware**

### ➤ Khái niệm:

* **Middleware** là các hàm trung gian xử lý action **trước khi** action đến reducer.

### ➤ Mục đích:

* Mở rộng chức năng của Redux, ví dụ:
  + Gọi API (bất đồng bộ).
  + Ghi log.
  + Kiểm tra quyền, xác thực.
  + Chặn hoặc thay đổi action.

### ➤ Một số middleware phổ biến:

* redux-thunk – xử lý async với hàm.
* redux-saga – xử lý async với generator.