SESSION 08:

Redux

Module: Training Program Preparation

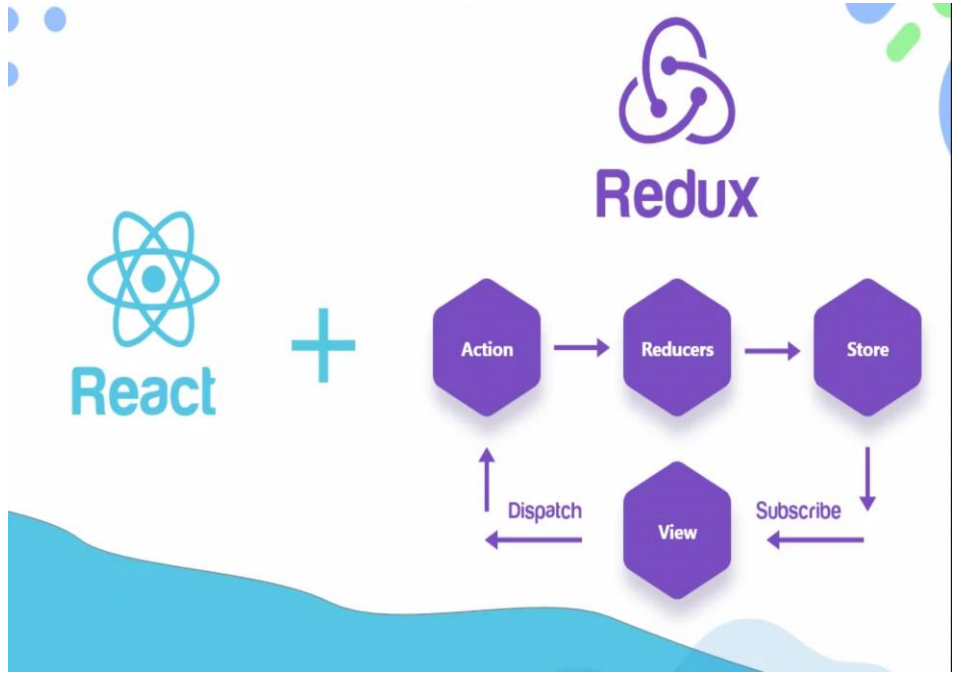
Version: 2.0

NỘI DUNG

1. Tổng quan Redux
2. Nguyên lý hoạt động của Redux
3. Các thành phần chính trong Redux
4. useSelector và useDispatch Hook
5. Redux Thunk
6. Redux Saga
7. So Sánh Redux Thunk với Redux Saga

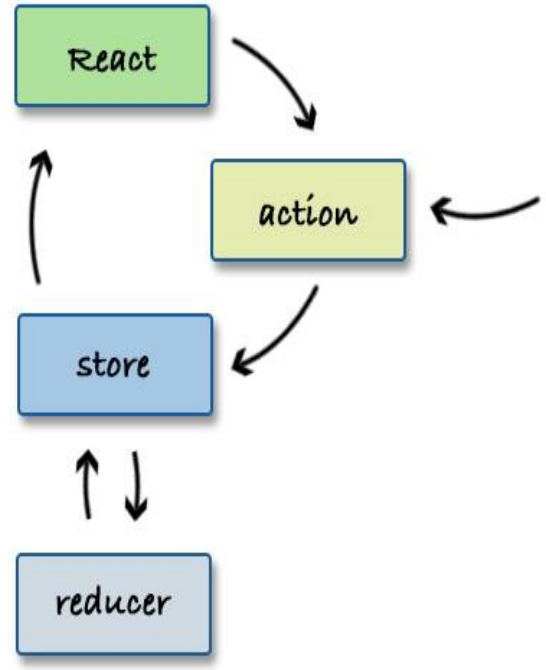
1. Tổng quan Redux - 1

- **Redux là gì?**
  - **Predictable state management tool** (công cụ quản lý trạng thái) dành riêng cho các ứng dụng của Javascrip
  - Thư viện Javascript dùng để quản lý và cập nhật trạng thái của ứng dụng
  - Giúp viết các ứng dụng hoạt động một cách nhất quántrong các môi trường khác nhau (client, server, and native) và dễ dàng kiểm thử.
  - **Redux** ra đời lấy cảm hứng từ tư tưởng của ngôn ngữ **Elm** (ngôn ngữ lập trình phản ứng chức năng) và kiến trúc **Flux** của Facebook. Do vậy Redux thường dùng kết hợp với React.
  - *Tài liệu tham khảo: https://redux.js.org/*



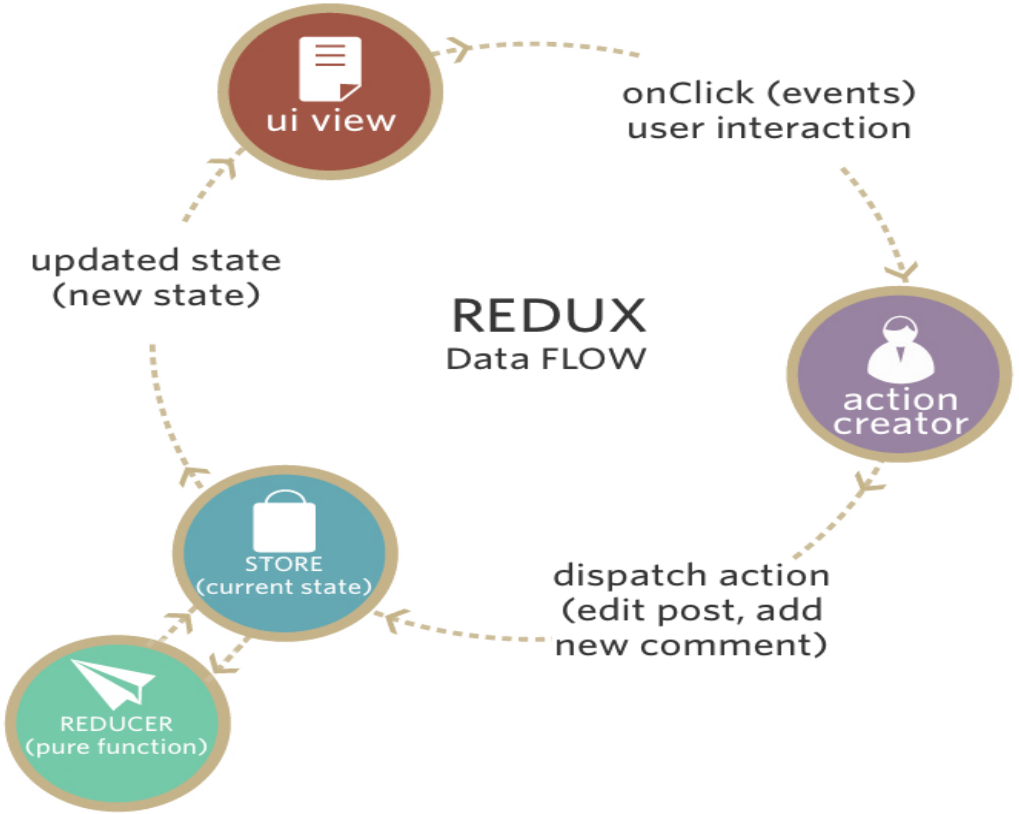
1. Tổng quan Redux - 2

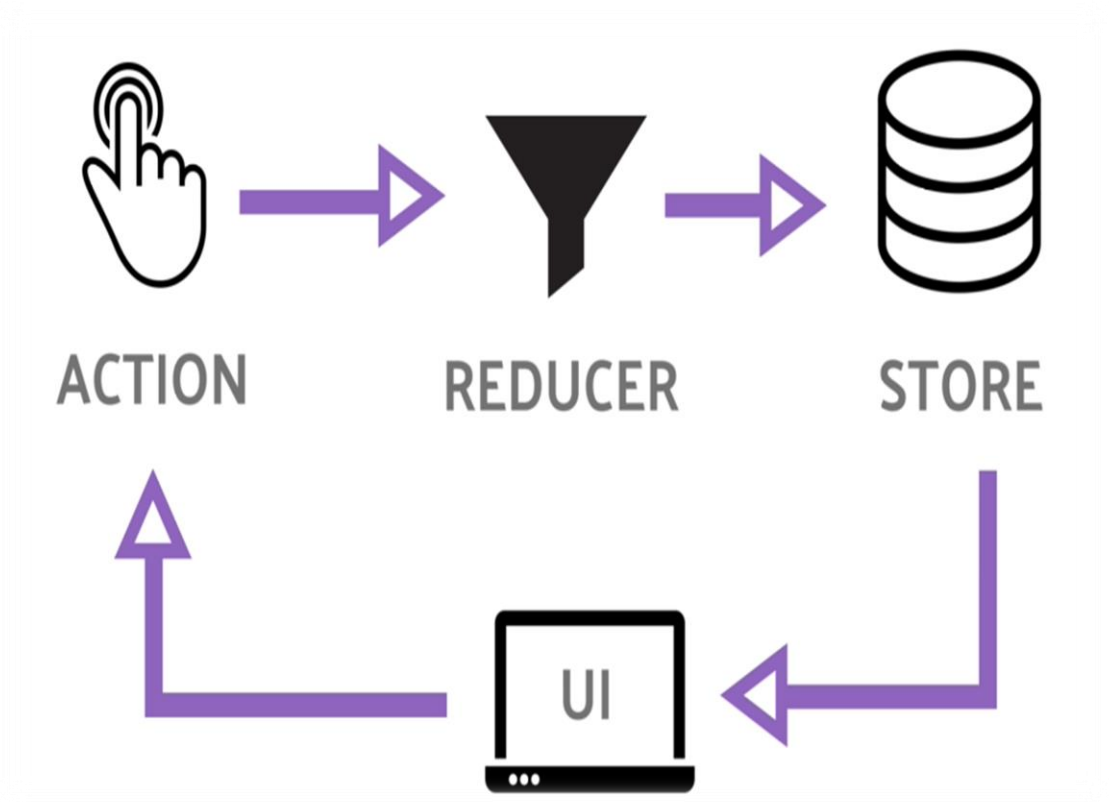
- **Redux xây dựng dựa trên 3 nguyên lý**
  - **Nguồn dữ liệu tin cậy duy nhất:** State của toàn bộ ứng dụng được chứa trong một object tree nằm trong **Store** duy nhất
  - **Trạng thái chỉ được phép đọc:** Cách duy nhất để thay đổi State của ứng dụng là phát một **Action** – object mô tả những gì xảy ra
  - **Thay đổi bằng hàm thuần túy:** Để chỉ ra cách mà State được thay đổi bởi Action phải dùng pure function gọi là **Reducer**



1. Tổng quan Redux - 3

- **Redux Flow**

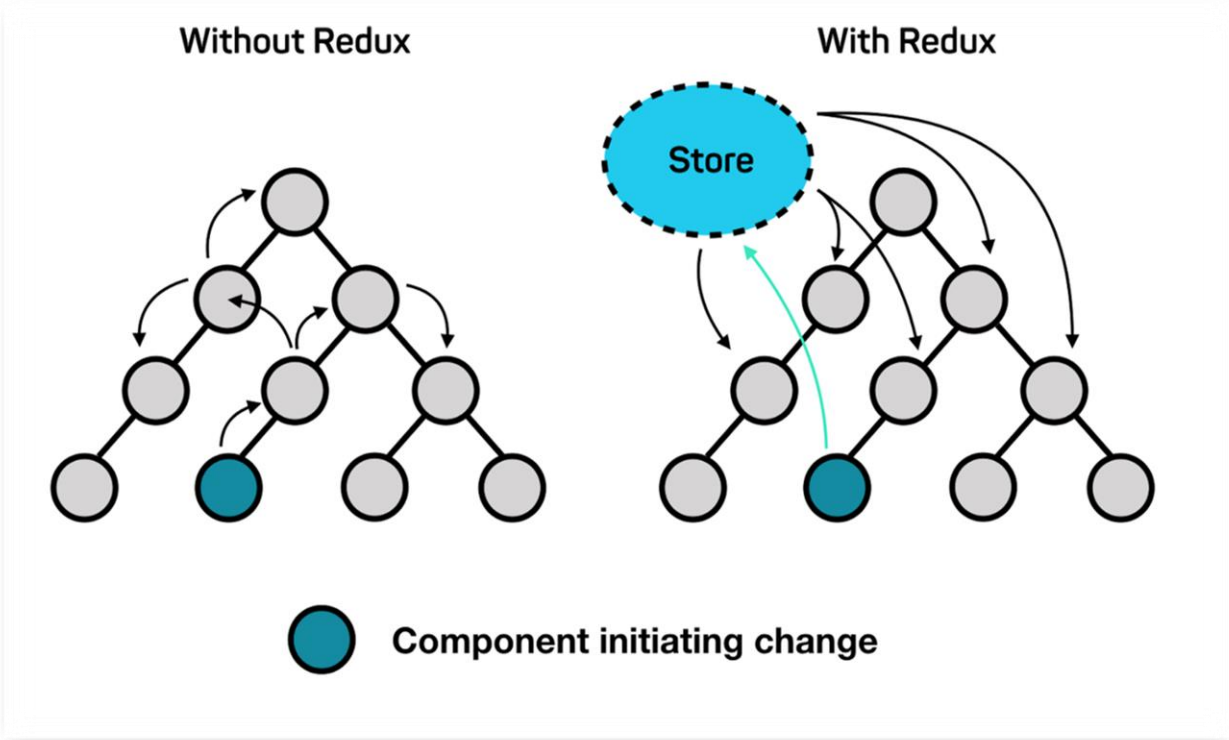




1. Tổng quan Redux - 4

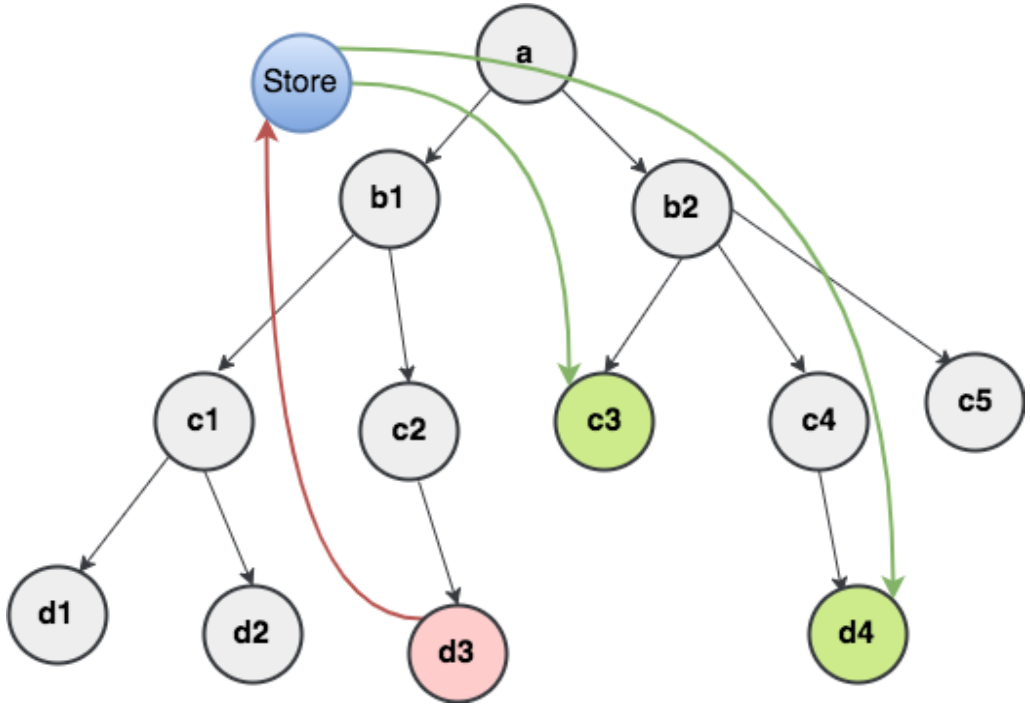
- **Tại sao nên dùng Redux?**
  - Thay đổi cách chia sẻ dữ liệu giữa các Component giúp giảm sự phức tạp
  - Giúp dự đoán được trạng thái State
  - Khả năng bảo trì
  - Gỡ lỗi nhanh chóng và dễ dàng
  - Lợi thế về hiệu suất
  - Tính bền bỉ
- **Cài đặt Redux**

```
npm install --save redux
npm install --save react-redux
```

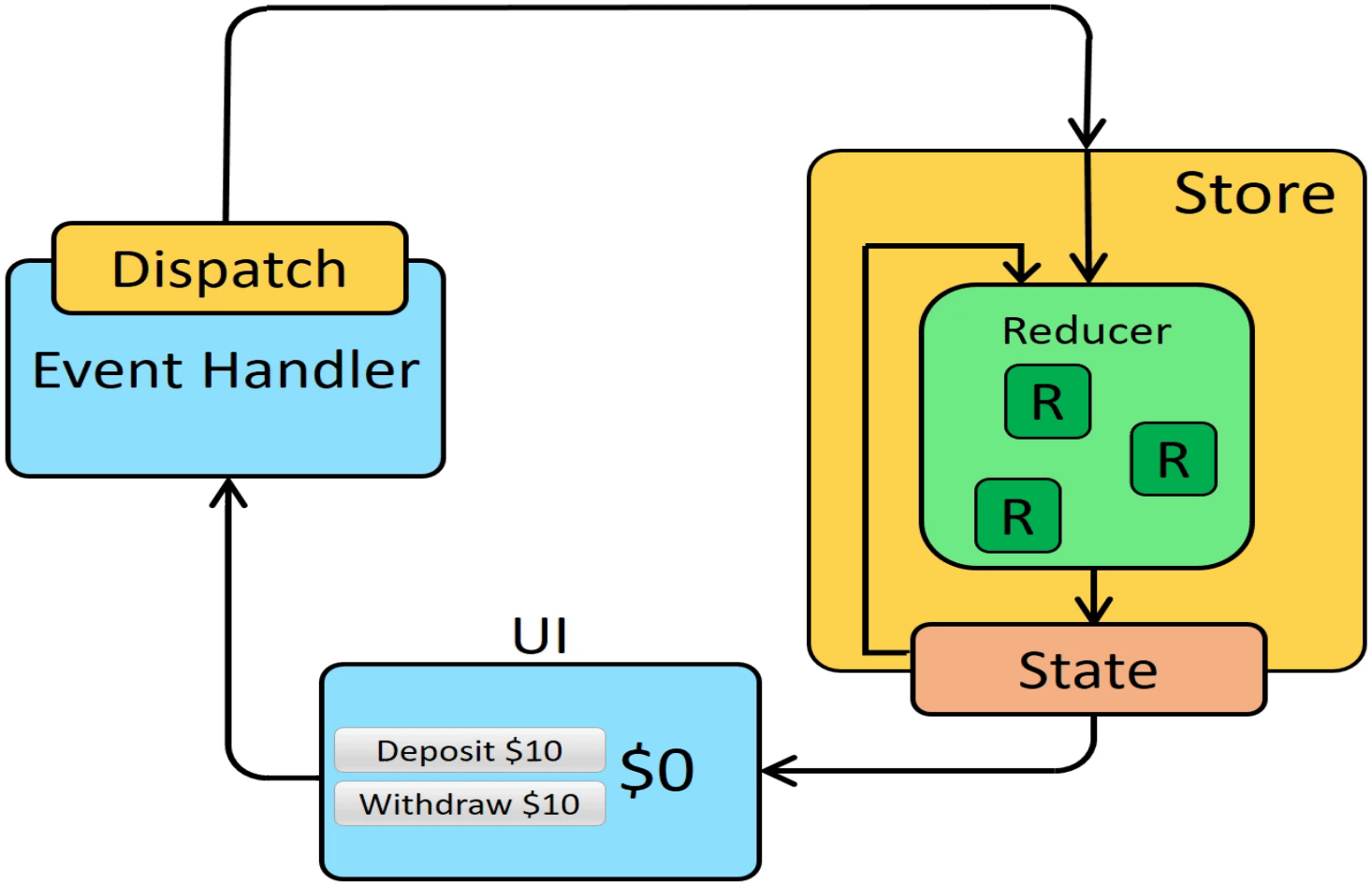


2. Nguyên lý hoạt động của Redux - 1

- **Store** lưu trữ toàn bộ **state** của ứng dụng
  - Các **component** có thể **access** trực tiếp đến **state** được lưu trữ
  - Nơi thay đổi state cũ thành state mới và cập nhật ra view
- **Action** là javascript object xử lý logic
- **View** là giao diện người dùng

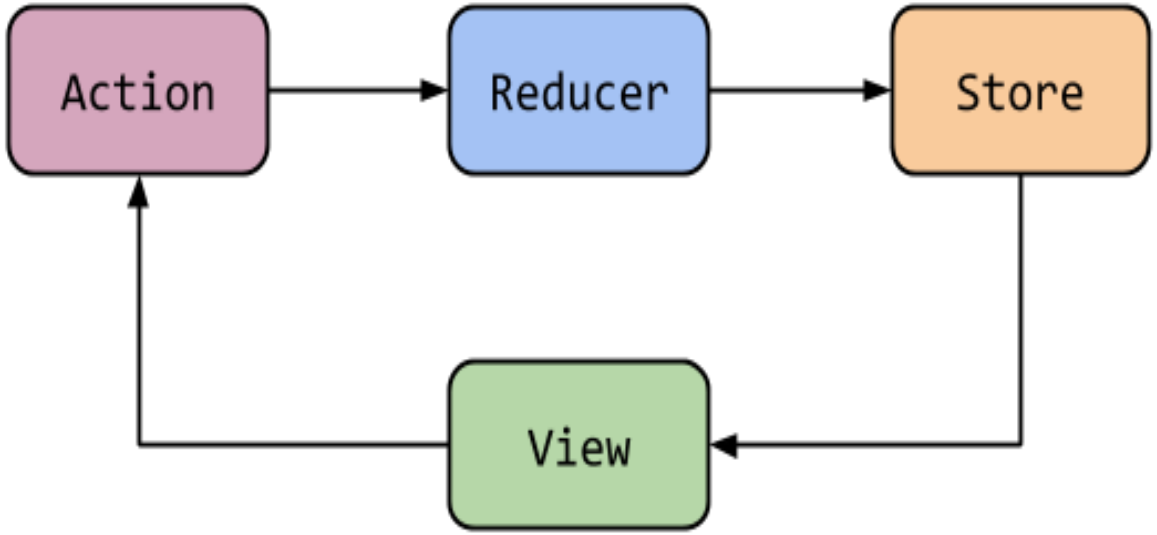


2. Nguyên lý hoạt động của Redux - 2



3. Các thành phần chính trong Redux - 1

- Redux sẽ hoạt động dựa vào 4 thành phần cơ bản là: **Actions**, **Reducers** , **Store** và **View**



3. Các thành phần chính trong Redux - 2

- **Actions**
  - Là các **events** được người dùng sử dụng để gửi dữ liệu từ ứng dụng đến **Redux Store**. Những dữ liệu này có thể có từ sự tương tác của user với app, API calls hoặc cũng có thể là từ form submission.
  - Là đối tượng được gửi bằng cách sử dụng **store.dispatch()** method
    - Phải có một **type property** để biểu lộ loại action để thực hiện.
    - Phải có một **payload** chứa thông tin
  - Được tạo thông qua một **action creator** .

```
export const act_count_up = (value) => {
    return {
        type: "COUNT_UP",
        payload: value
    }
};
```

3. Các thành phần chính trong Redux - 3

- **Reducer**
  - Là các function nguyên thủy lấy state hiện tại của ứng dụng, thực hiện một action và trả về một state mới. Gồm 2 tham số
    - State: state hiện tại của ứng dụng
    - Action: action được gửi thông qua dispatch
  - Những state này được lưu như những objects và chúng định rõ cách state của một ứng dụng thay đổi trong việc phản hồi một action được gửi đến store

```
const reducer = (state, action) => {...}
```

3. Các thành phần chính trong Redux - 4

- **Reducer**

```
const initialState = 0;
const count = (state = initialState, action) => {
    console.log("vao reducer");
    switch (action.type) {
        case "COUNT_UP":
            state += action.payload;
            return state;
        case "COUNT_DOWN":
            state -= action.payload;
            return state;
        default:
            return state;
    }
}
export default count;
```

```
const initialState = [1, 3, 5, 7];
const listNumber = (state = initialState, action) => {
    switch (action.type) {
        case "RANDOM":
            state = [...state, parseInt(Math.random() * 10)];
            return state;
        default:
            return state;
    }
}
export default listNumber;
```

```
import { combineReducers } from 'redux';
import count from './count';
import listNumber from './listNumber';
export const reducer = combineReducers({ count, listNumber });
```

3. Các thành phần chính trong Redux - 5

- **Store**
  - Nơi quản lý trạng thái (state) , có thể truy cập để lấy trạng thái (state) ra , update state hiện có và lắng nghe để nhận biết xem có sự thay đổi nào không và cập nhật nó qua views
  - Trong store có 3 phương thức:
    - **getState ()** : Giúp lấy ra state hiện tại
    - **dispatch(action)** : thực hiện gọi 1 action
    - **subscribible(listener)** : luôn lắng nghe sự thay đổi của state rồi ngay lập tức sẽ cập nhật ra **Views**
  - Để sử dụng được **Store** thì cần phải **import createStore** từ thư viện **redux**

```
import { createStore } from "redux";
import reducer from "./reducer";

export const store = createStore(reducer);
```

```
import { Provider } from 'react-redux';
root.render(
  <Provider store={store}>
    <App />
  </Provider>
);
```

4. useSelector và useDispatch hooks - 1

- **useSelector ()**
  - **useSelector ()** là một hook cho phép chúng ta lấy **State** từ **Redux store** bằng cách sử dụng một **selector function** là tham số đầu vào .
  - **useSelector** sẽ return về bất cứ State nào mà bạn muốn
  - Cơ chế của **useSelector** là so sánh giá trị state trước và sau nếu khác nhau thì component sẽ tự động re-render lại , ngược lại giống nhau sẽ không re-render lại component

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

```
const count = useSelector(state => state.count)
return (
    <div>
        <p>Current Value: {count}</p>
    </div>
)
```

4. useSelector và useDispatch hooks - 2

- **useDispatch ()**
  - **useDispatch ():** return về một tham chiếu đến **dispatch function** từ **redux store** và được sử dụng để **dispatch** các **action**

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

```
const dispatch = useDispatch();
const handleUp = (value) => {
    dispatch(act_count_up(value));
}
return (
    <div>
        <h2>Redux Demo</h2>
        <button onClick={() => handleUp(3)}>UP</button>
        <button onClick={() => dispatch(act_count_down(2))}>DOWN</button>
    </div>
)
```

3. Redux Thunk

- **Redux Thunk** là một phần mềm trung gian (**middleware**) cho phép trả về các **function** thay vì chỉ là **action** trong **Redux**.
- Một trong những ứng dụng chính của **Redux Thunk** là dùng để xử lý **action** không đồng bộ, chẳng hạn như sử dụng axios để gửi một GET request.

3. Redux Thunk

- **thunkFunction** là một hàm chấp nhận hai đối số: phương thức lưu trữ **Redux dispatch** và phương thức lưu trữ **Redux getState**.
- **thunkFunction** không được gọi trực tiếp bằng mã ứng dụng. Thay vào đó, chúng được chuyển đến **store.dispatch()**:
- **thunkFunction** có thể chứa bất kỳ logic tùy ý, đồng bộ hóa hoặc không đồng bộ nào và có thể gọi **dispatch** hoặc **getState** bất kỳ lúc nào.

```
const thunkFunction = (dispatch, getState) => {
  // logic here that can dispatch actions or read state
}

store.dispatch(thunkFunction)
```

3. Redux Saga

- **Redux-Saga** là một thư viện **redux middleware**, giúp quản lý những **side effect** trong ứng dụng **redux** trở nên đơn giản hơn.
- Bằng việc sử dụng tối đa tính năng **Generators** (function*) của ES6, nó cho phép ta viết async code nhìn giống như là **synchronos**.

3. So sánh Redux Thunk với Redux Saga

	Redux Thunk	Redux-saga
Ưu điểm	Đơn giản, mạnh mẽ, dễ sử dụng, dễ tiếp cận đối với các bạn là mới học React	Đối với những dự án phức tạp sử dụng redux-saga code sẽ clean và dễ test hơn so với redux-thunk, giải quyết được những vấn đề về promises
Nhược điểm	Chỉ phù hợp với các dự án nhỏ, xử lý logic đơn giản. Còn đối với những dự án phức tạp sử dụng redux-thunk sẽ phải tốn nhiều dòng code và gây khó khăn cho việc test các action	Phức tạp, tốn thời gian cho member mới, nặng về xử lý logic

TỔNG KẾT

- ❑ Tổng quan Redux
- ❑ Nguyên lý hoạt động của Redux
- ❑ Các thành phần chính trong Redux
- ❑ useSelector và useDispatch Hook
- ❑ Redux Thunk
- ❑ Redux Saga
- ❑ So Sánh Redux Thunk với Redux Saga

RIKEI Academy®
where the dream begins

KẾT THÚC

HỌC VIỆN ĐÀO TẠO LẬP TRÌNH CHẤT LƯỢNG NHẬT BẢN