ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
VNUHCM - UIT

Khoa: **Công Nghệ Phần Mềm**
Lớp: **SE347.P11**

BÁO CÁO SEMINAR
CÔNG NGHỆ WEB VÀ ỨNG DỤNG

STATE MANAGEMENT
REDUX & REDUX SAGA

GVHD: ThS. **Trần Thị Hồng Yến**

SVTH1: **Nguyễn Hoàng Duy– 22520328**

SVTH2: **Nguyễn Sơn Bình– 22520135**

State Management

Redux & Redux Saga

Contents

01

State Management

Tại sao chúng ta cần quản lý state?

02

Redux

Redux là gì? Ưu điểm và nhược điểm? Kiến trúc redux?

03

Redux Saga

Redux Saga là gì? Tại sao cần sử dụng redux saga?

04

Demo

Demo thực tế với redux & redux saga trong quản lý state

01
State Management

State và Props là gì?

State

- Là một đối tượng trong component, có thể thay đổi và quản lý trạng thái của component.
- Giống như kho lưu trữ dữ liệu, dùng để cập nhật component khi người dùng thực hiện hành động như nhấp nút, nhập văn bản, v.v.
- Khi state thay đổi, component tự động re-render để cập nhật giao diện.
- Chỉ tồn tại trong component mà nó được định nghĩa, không thể truy cập trực tiếp từ component khác.

Props

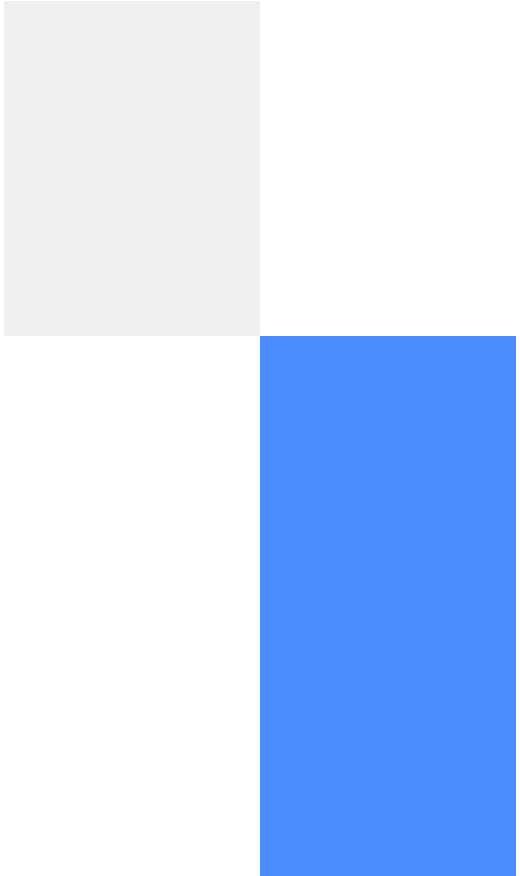
- Viết tắt của "properties", là các tham số được dùng để truyền dữ liệu và các phương thức từ component cha xuống component con.
- Mọi component được coi là một hàm javascript thuần khiết (Pure Function).
- Props tương đương với các tham số của hàm javascript thuần khiết.
- Props là bất biến (immutable).

Các vấn đề khi quản lý state

- **State không đồng nhất:** Các thành phần có thể có trạng thái khác nhau, dẫn đến hành vi không nhất quán.
- **Khó khăn trong việc theo dõi dữ liệu:** Phức tạp khi cần chia sẻ và cập nhật dữ liệu giữa nhiều thành phần.
- **Tăng độ phức tạp của mã nguồn:** Mã nguồn trở nên khó đọc và bảo trì.
- **Khó khăn trong việc debug:** Thiếu công cụ theo dõi sự thay đổi của state, gây khó khăn trong việc tìm lỗi.
- **Giảm hiệu suất:** render lại không cần thiết của các thành phần, làm giảm hiệu suất của ứng dụng.
- **Khả năng mở rộng kém:** Quản lý state trở nên phức tạp khi ứng dụng lớn lên.
- **Khó khăn trong việc test:** Việc viết unit test và integration test trở nên khó khăn.

02

Redux



Redux là gì?

Redux là một **predictable state management tool** được dành cho các ứng dụng JavaScript. Nó giúp viết các ứng dụng hoạt động một cách nhất quán, chạy trong nhiều môi trường khác nhau (client, server, và native), và dễ dàng kiểm thử.

***Fact:** Redux ra đời lấy cảm hứng từ tư tưởng của ngôn ngữ Elm và kiến trúc Flux của Facebook. Do vậy Redux thường dùng kết hợp với React.

Redux hoạt động dựa trên một store duy nhất lưu trữ toàn bộ state của ứng dụng. Các component có thể truy cập trực tiếp state này thay vì phải truyền props từ component cha xuống component con.

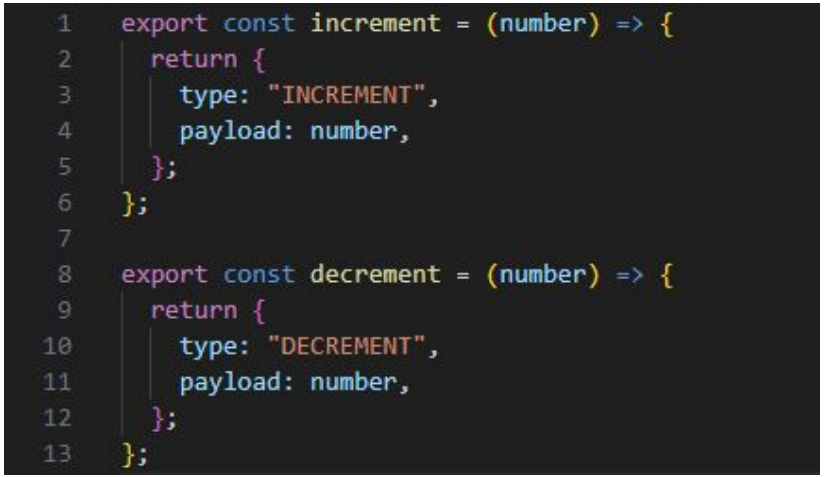
Redux bao gồm ba thành phần chính:

- **Actions**: Các đối tượng mô tả hành động hoặc sự kiện xảy ra trong ứng dụng.
- **Store**: Nơi lưu trữ toàn bộ state của ứng dụng.
- **Reducers**: Các hàm xác định cách state thay đổi dựa trên các actions.

Action

Actions trong Redux đơn giản là các sự kiện gửi dữ liệu từ ứng dụng đến Redux store. Dữ liệu này có thể đến từ tương tác người dùng, các cuộc gọi API, hoặc form submission. Khi định nghĩa actions, chúng ta khai báo tên các hành động trong ứng dụng.

★ Ví dụ, với một state là **counter**, chúng ta cần hai phương thức để tăng và giảm giá trị của counter, do đó, ta định nghĩa hai actions là '**INCREMENT**' và '**DECREMENT**'. Việc xử lý thay đổi state sẽ được đảm nhiệm bởi reducer.



```
export const increment = (number) => {
  return {
    type: "INCREMENT",
    payload: number,
  };
};

export const decrement = (number) => {
  return {
    type: "DECREMENT",
    payload: number,
  };
};
```

Reducer

Reducers là các hàm xử lý các hành động và thay đổi state của ứng dụng. Chúng nhận vào state hiện tại và một action, sau đó trả về state mới dựa trên action đó. Mỗi reducer tương ứng với một phần của state và mô tả cách state thay đổi khi các action khác nhau được gọi.

★ Ví dụ, reducer **counter** lưu state của counter, kiểm tra action là '**INCREMENT**' hay '**DECREMENT**', và trả về state mới là state +number hoặc state -number tương ứng.

```
const counterReducer = (state = 0, action) => {
  switch (action.type) {
    case "INCREMENT":
      return state + action.payload;
    case "DECREMENT":
      return state - action.payload;
    default:
      return state;
  }
};

export default counterReducer;
```

```
import { combineReducers } from "redux";

import counter from "./counter";

const allReducers = combineReducers({
  counter,
  // add more reducers here
});
```

Store

Store trong Redux đơn giản là một đối tượng chứa toàn bộ state toàn cục của ứng dụng. Thay vì lưu trữ trực tiếp các state, nó lưu các reducer, giống như một kho chứa. Khi một component cần sử dụng state nào đó, nó có thể truy cập trực tiếp vào store để lấy ra.

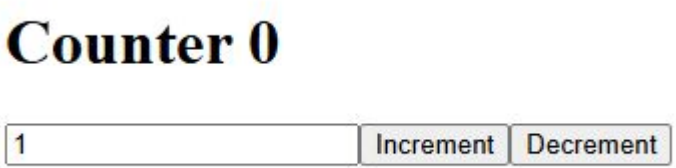
★ Ví dụ, có counter reducer rồi, ta bỏ nó vào **store**. Redux hỗ trợ phương thức **createStore** nhận vào reducer và trả về **store**:

```
1  import React from "react";
2  import ReactDOM from "react-dom";
3  import { createStore } from "redux";
4  import { Provider } from "react-redux";
5
6  import App from "./App";
7  import allReducers from "./reducers";
8
9  const store = createStore(allReducers);
10
11 ReactDOM.render(
12   <Provider store={store}>
13     <App />
14   </Provider>,
15   document.getElementById("root")
16 );
```

Các Dispatch

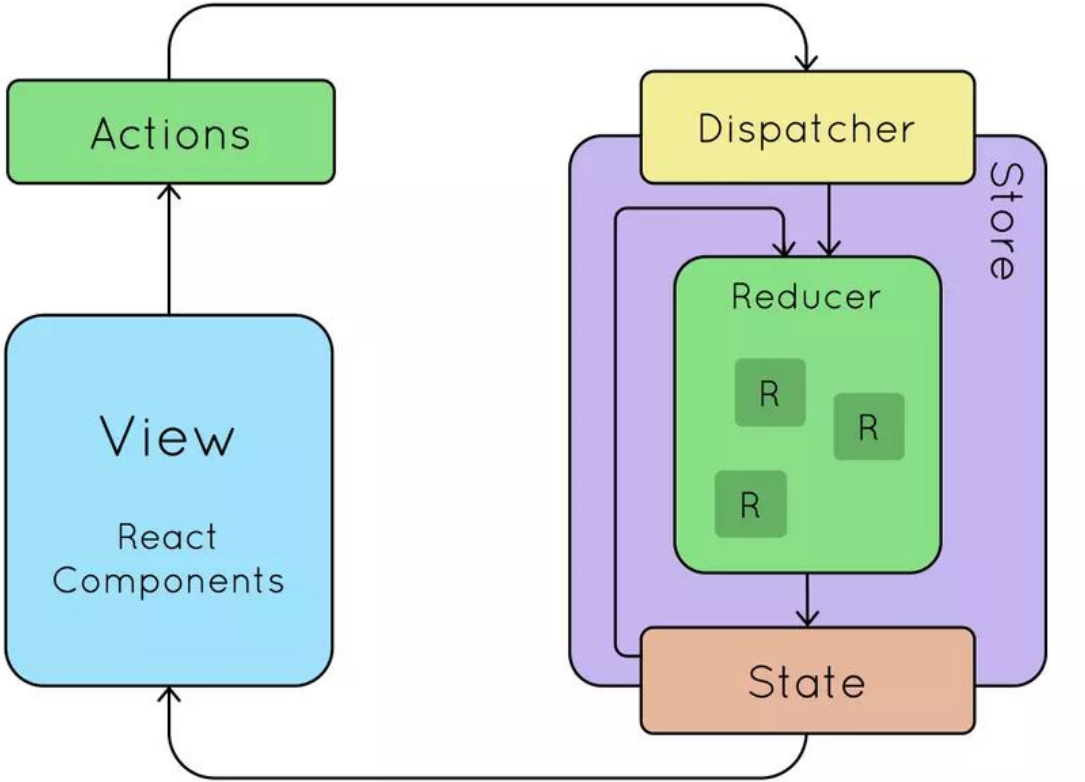
Các Dispatch: Khi cần dùng 1 action ở component, ta gọi action đó đơn giản bằng cách sử dụng phương thức **dispatch**.

★ VD:**dispatch(increment()), dispatch(decrement()).**




```jsx
import React, { useState } from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { increment, decrement } from './actions/counterActions';

function App() {
  const counter = useSelector((state) => state.counter);
  const dispatch = useDispatch();

  const [value, setValue] = useState(0);
  const [amount, setAmount] = useState(1);

  const handleValueChange = (e) => {
    setValue(parseInt(e.target.value));
  };

  const handleAmountChange = (e) => {
    setAmount(parseInt(e.target.value));
  };

  return (
    <div>
      <h1>Counter {counter}</h1>
      <input
        type="number"
        value={amount}
        onChange={handleAmountChange}
        placeholder="Enter amount"
      />
      <button onClick={() => dispatch(increment(amount))}>Increment</button>
      <button onClick={() => dispatch(decrement(amount))}>Decrement</button>
    </div>
  );
}

export default App;
```

Luồng xử lý của Redux



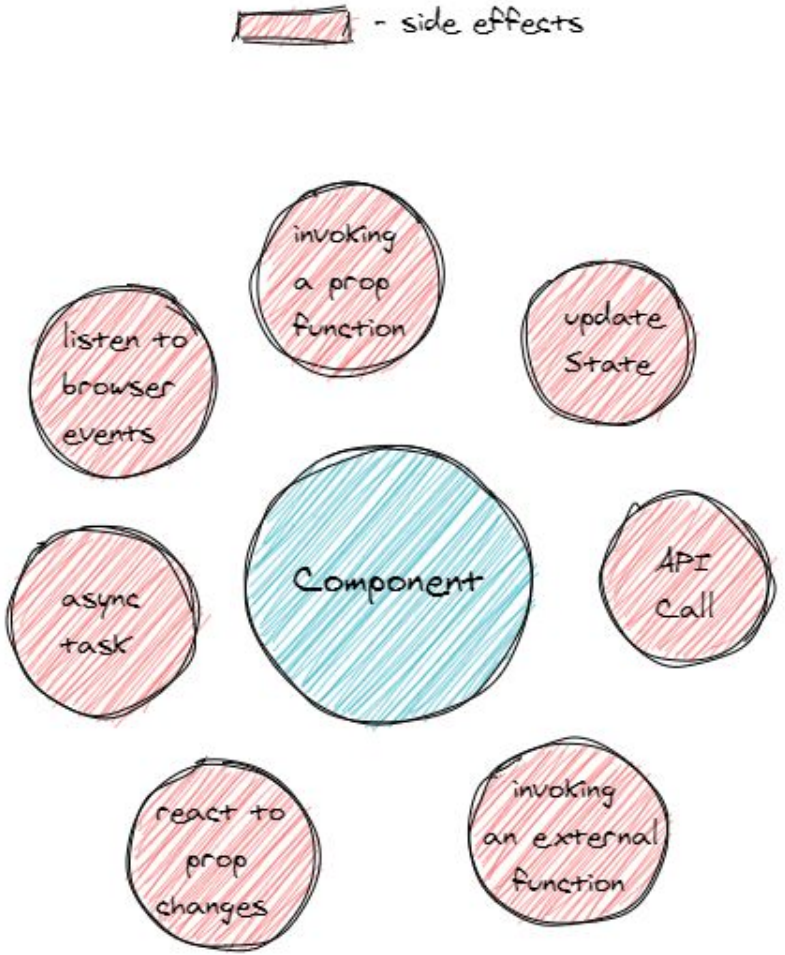
Ưu điểm và Nhược điểm

Ưu điểm	Nhược điểm
Quản lý trạng thái tập trung: Cung cấp một kho lưu trữ tập trung cho quản lý state, giúp các thành phần đồng bộ và dễ dàng gỡ lỗi.	**Độ phức tạp cao**: Redux có rất nhiều chức năng. Đồng nghĩa với việc nó sẽ có hệ thống vận hành tương đối phức tạp.
Luồng dữ liệu dự đoán được: Luồng dữ liệu đơn hướng nghiêm ngặt, giúp dễ hiểu cách dữ liệu di chuyển và cải thiện độ ổn định của ứng dụng.	**Mã mẫu (Boilerplate Code)**: Yêu cầu viết nhiều mã mẫu để thiết lập store, actions và reducers, có thể tốn thời gian.
Tích hợp dễ dàng với React: ó API nhỏ và đơn giản, dễ học và sử dụng, hỗ trợ bởi thư viện React-Redux để kết nối với store.	**Hiệu suất:** Có thể thêm gánh nặng hiệu suất, đặc biệt với lượng dữ liệu lớn do yêu cầu sao chép state mỗi khi cập nhật.
Hỗ trợ cộng đồng lớn: Có cộng đồng lớn và năng động, nhiều tài nguyên học tập và hỗ trợ.	**Quá mức cần thiết cho các ứng dụng đơn giản**: Có thể quá mức cần thiết cho các ứng dụng đơn giản không có luồng dữ liệu phức tạp.

Redux Saga

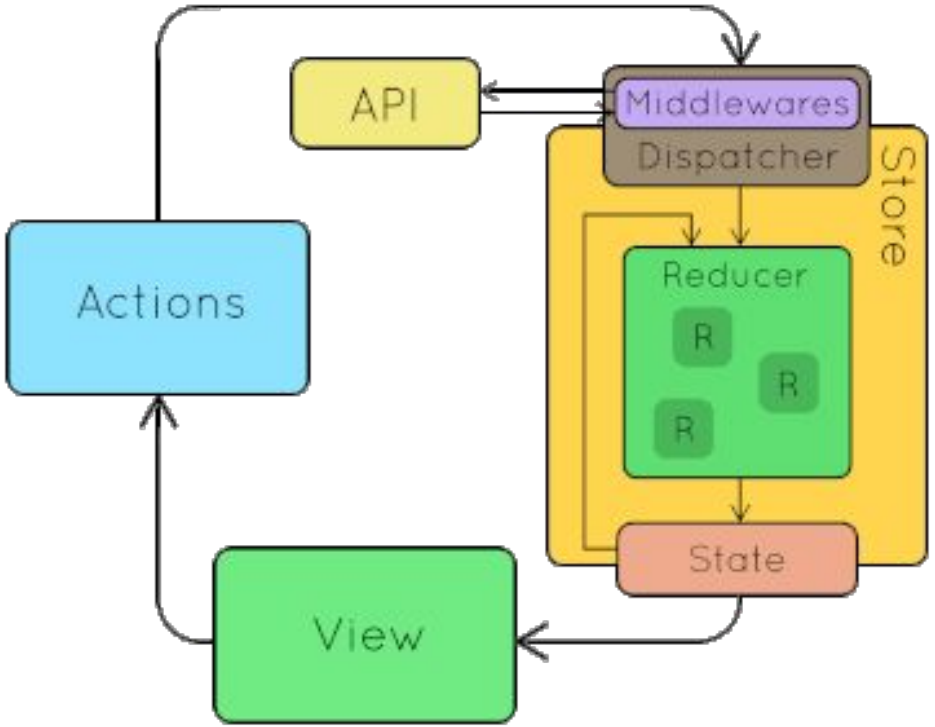
Side-Effect

“In React, “side effects” refer to any operations or behaviors that occur in a component after rendering, and that don't directly impact the current component render cycle. “

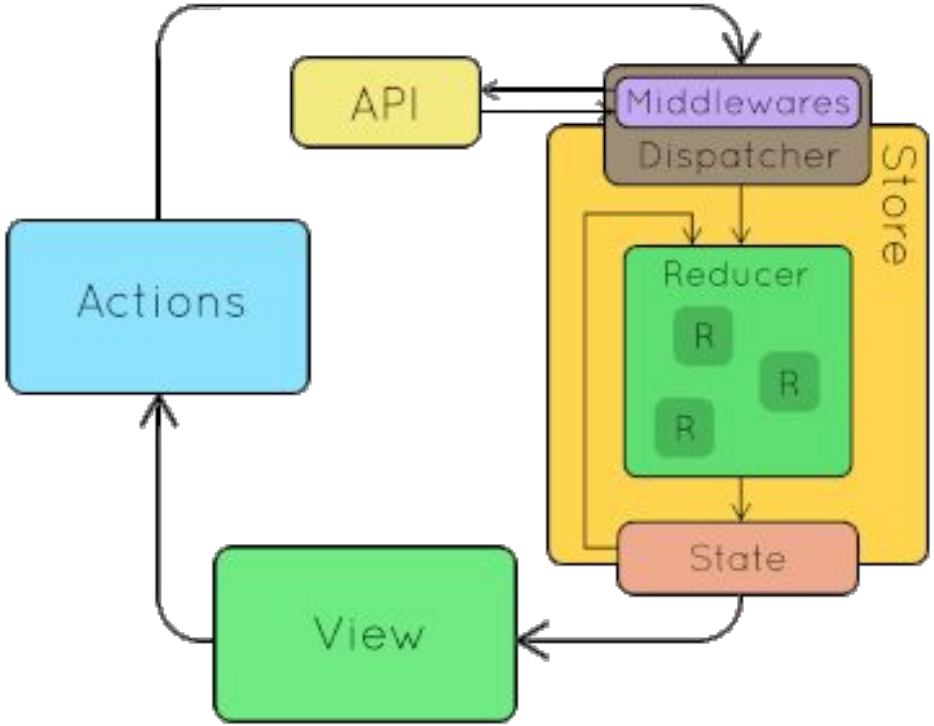


Redux Saga

Redux saga là một thư viện redux middleware, giúp quản lý những side effect trong ứng dụng redux trở nên đơn giản hơn.



Cách hoạt động của Redux Saga



1. Khi view dispatch 1 action lên cho reducer xử lý thì trước tiên nó phải đi qua middleware Saga
2. Khi Saga nhận được action mà view dispatch lên thì nó sẽ bắt lấy action đấy để xử lý
3. Sau khi saga xử lý xong thì nó sẽ dùng hàm *put* để dispatch một action mới lên cho reducer (action này có thể kèm theo cả dữ liệu mà saga đã xử lý trước đó)
4. Bây giờ thì reducer mới nhận được action, sau đó reducer sẽ xử các action theo các điều kiện khác nhau(tùy theo action mà saga gửi lên thì reducer xử lý).

Demo

Basic Redux & Saga implementation
https://github.com/BinhNguyen215/redux-app

Reference

- Redux - A Predictable State Container for JS Apps - https://redux.js.org/
- React Redux - Official React bindings for Redux - https://react-redux.js.org/
- Dev Ed - Redux For Beginners - https://www.youtube.com/watch?v=CVpUuw9XSjY

Bảng phân công

Thành viên	Công việc	Đánh giá
Nguyễn Hoàng Duy-22520328	Redux Saga Code Demo Trình bày demo	100%
Nguyễn Sơn Bình-22520135	Đặt vấn đề: State management Redux Code Demo	100%

Thanks

Do you have any questions?