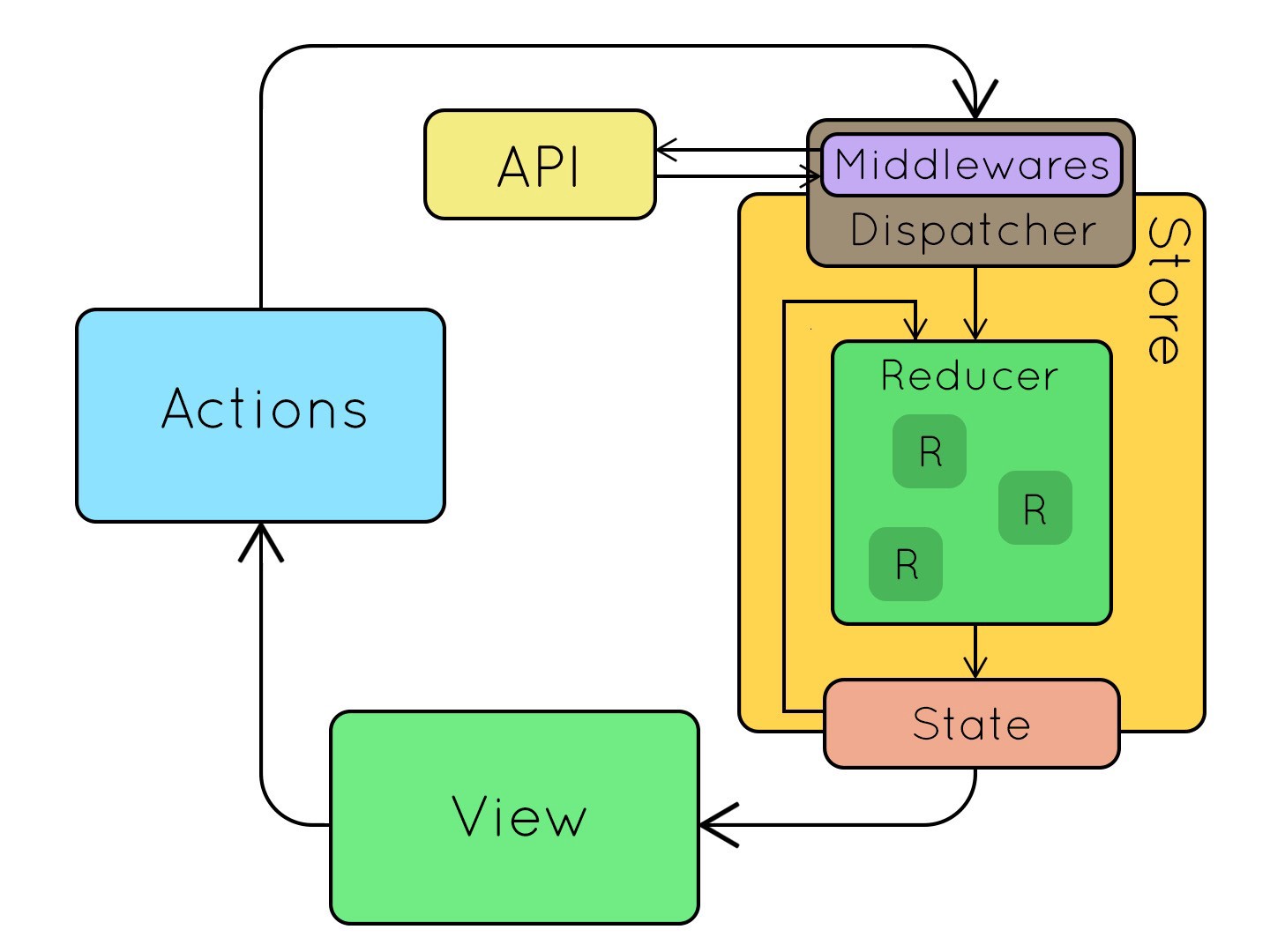
**REDUX-THUNK AND REDUX-SAGA**

****

**Cấu trúc của Redux:**

**Store :** Là **1 object** lưu trữ **State** của toàn bộ ứng dụng, cho phép truy cập **State** qua **getState()**, **update State** qua **Dispatch( action ).**

**Action Creators :** Là nơi tạo ra các **action** dùng để **mô tả event** do người dùng tạo ra.

**Reducer :** Là **1** **function** nhận đầu vào là state và **các mô tả event** và dựa trên đó để trả về **state** tiếp theo nhưng không thay đổi state cũ.

**View :** Hiển thị dữ liệu được cung cấp bởi **store**.

**Nguyên tắc trong Redux:**

**Redux:** Sử dụng kiến trúc **1 chiều** , Các **function Reducer** phải là **Pure function** (với cùng 1 đầu vào chỉ cho ra 1 đầu ra duy nhất)

**Store :** Chỉ dùng **1 store** duy nhất trong app, luôn là nguồn dữ liệu **đúng** và **tin cậy** duy nhất.

**State :** Chỉ được phép đọc, cách duy nhất để thay đổi **State** là phát sinh một **Action**, và để **Reducer** thay đổi **State**.

**Redux-Thunk:**

Là một **package**. Cung cấp **middleware** để sử lý các hàm trả về từ **action creator**. **Middleware** và **action creator** làm việc cùng nhau.

Dùng để sử lý **side-effect** và các hoạt động không đồng bộ ( **asynchronous** ).

Thân thiện, dễ sử dụng, mạnh mẽ, giải quyết hầu hết các vấn đề.

**Điểm yếu :** Khó đọc, khó maintain.

**Side-Effect:** Là những tương tác của ứng dụng với thế giới bên ngoài: giao tiếp **API**, đọc ghi **file**, **analyst events.**

**Trong redux:** **side-effect** thường được xử lý ở **action** ( **action creator** ) hoặc **middleware.**

* **Lưu ý:** không sử lý ở **reducers**, vì **reducer** phải là **pure function** – hàm thuần túy.

**Middleware:**

Phân chia ứng dụng của bạn thành các lớp **middleware** khác nhau 🡺 **request** của người dùng sẽ đi qua các lớp **middleware** này. Ví dụ: kiểm tra quyền , kiểm tra dữ liệu, ...

**Middleware** : là lớp nằm ở giữa **Reducers** và **Dispatch** **Actions.**

**Vị trí hoạt động:**

+ Trước khi **reducers** nhận được **action**.

+ Sau khi **action** đã đươc **dispatch**.

Thường được sử lý **Async Action**.

**Async Action :**

+ Là action chưa sẵn sàng khi được gọi.

Ví dụ : sử dụng **redux-thunk** để trì hoãn việc **dispatch** 🡺 **fetch** dữ liệu xong mới **dispatch**.

**Redux-saga:**

Thay thế chỗ cho **Redux-Thunk** . Ngoài ra chúng ta có thể sử dụng **Redux-saga** và **Redux-Thunk** cùng một lúc.

+ Xử lý được **side-effect** .

+ Xử lý các câu chuyện phức tạp về dữ liệu.

+ Giao tiếp với thế giới bên ngoài : viết các chức năng **real time**, xây dựng các ứng dụng **chat**.

+ Sử dụng tính năng **ES6** : **generators**.

+ Có khả năng chuyển code không đồng bộ thành đồng bộ.

**Điểm mạnh:**

+ Xử lý các quy trình dài hạn, phức tạp.

+ Giải quyết các bài toán về **chains of promises**.

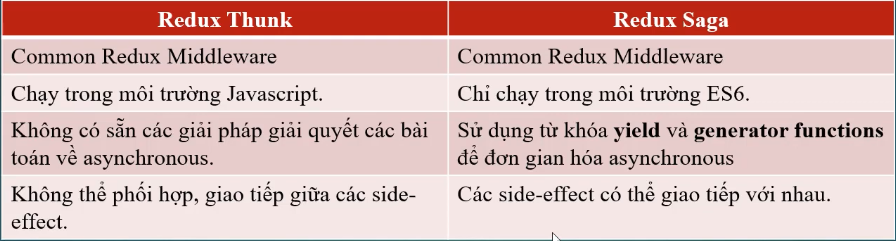
+ Giúp **clean code**.

**Điểm yếu:**

+ Phức tạp.

+ Tốn thời gian dành cho thành viên mới vào **team**.

+ Nặng về sử lý. Không dành cho ứng dụng đơn giản.



**Saga effect :**

+ Quản lý **effect** đựa vào các **helper** : **Fork**, **take**, **call**, ...

+ Saga trả về các mô tả logic ở dạng đối tượng, hay còn gọi là **effect**. **Effect** sử dụng các **helper methods** để trả về các effect. Sau đó gửa cho **saga middleware** xử lý các **effect**.

**Method Helper: Fork :** Thuộc về redux - saga/effects

+ Đầu vào của **fork** là 1 **generator function**.

+ Dùng để rẽ nhánh để sử lý công việc , như **if, else, switch, case**

+ Người theo dõi **action**.

+ Mỗi **fork** là **: non-blocking**, có nghĩa là không chờ đợi ai hết việc ai nấy làm. **Or** có thể hiểu là có thể kích hoạt nhiều bộ theo dõi cùng một lúc. ( **dispatch** nhiều **action** cùng một lúc ).

**Khi nào sử dụng** **fork** :

+ Nhóm các **saga** theo miền **logic** nào đó. VD: nhóm theo chức năng.

+ Lưu trữ tham chiếu đến các **sagas** để có thể hủy hoặc vào lại.

**Method Helper: Take**

+ Đầu vào của nó là một **action.** Dùng để phản hồi các **action** khi **action** được **dispatch** thực thi. **Theo dõi lần đầu thôi.** 🡺 để khắc phục sử dụng **vòng lặp vô tận**.

+ Lệnh **take** sẽ được kích hoạt và tham gia vào **saga** khi một **action** được **dispatch**. Tạm dừng đến khi nó nhận được một **action**.

+ là **Blocking** : những dòng lệnh phía dưới chỉ chạy khi **action** mà nó nhận vào được **dispatch**.

**Method Helper: Call**

+ Tương tự như **call** của javascript.

+ Thường được dùng để gửi **request API, call API.**

+ Là **blocking**.

+ giống như việc thực thi **1** **function**. Trả về **Promise** và sẽ tạm dừng **saga** cho đến khi **promise** được **resolved**.

**Method Helper: Put**

+ Dùng để **dispatch** một cái **action** trong **Saga**. Đầu vào là **một action**.

+ Là **non-blocking**.

**Method Helper: Delay**

+ Là một **blocking**.

+ Để chặn thực thi trong một khỏng thời gian là miliseconds ( đầu vào của **delay** ).

**Method Helper: TakeLatest**

+ Thay thế cho **fork**.

+ Không cần vòng lặp vô hạn.

+ Nếu thực hiện một loạt các **action, takeLates** chỉ thực thi và **lấy kết quả** của **action** cuối cùng.

+ Hủy bỏ một quy trình cũ khi có một quy trình mới bắt đầu.

+ Sử dụng nhiều trong chức năng **search** tìm kiếm.

Ở hàm **saga**: nó sẽ nhận được vào **action:**

function\* filterTaskSaga({ payload }) {}

**Method Helper: Select**

+ Lấy **data** từ **store** tại **saga**.

  const taskEditing = yield select((state) => state.tasks.taskEditing);

**Method Helper: TakeEvery**

+ Ngược lại với **TakeLatest**, **TakeEvery** sẽ chạy ngay lập tức nếu được kích hoạt, chạy không tính số lần, gọi là chạy, chạy không biết là **action** trước đó **đã chạy song chưa**.

**Method Helper: All**

+

**Tóm tắt:**

+ Điểm mạnh **redux-saga**: mô tả logic, quy trình thực hiện.

+ Khác biệt thunk xử lý khó khăn: hủy bỏ những yêu cầu cũ khi có **request** mới đến.

**Redux-form:**

Quản lý **state form** trên **redux**.

Hỗ trợ mạnh **validation**.

**This.props** hỗ trợ nhiều **state của redux-form** được chuyển đổi thành **props của component.**

**Validation : ở phía client**

Ràng buộc dữ liệu