### Redux-saga

### Generator là object được trả về từ generator function

### Generator không thể tạo ra trực tiếp

### Generator function có 3 hàm: next(),return(),throw()

### Generator function hàm có thể dừng giữa chừng và tiếp tục sau

### Generator function mỗi lần gọi, hàm sẽ được thực thi cho đến khi gặp yield và return

### Redux middleware: cung cấp một tầng đứng ở giữa việc dispatch action và gửi lên reducer. Thường được dùng trong việc logging, reporting, async api, routing, ...

### Effect là javascript obj chứa thông tin để middleware biết cần làm gì

### Effect Creator: là một function trả về một Effect. Và nó không thực thi Effect này, saga middleware, chứ ko phải Effect Creator nhé.

### Các hàm các bạn dùng trong Redux Saga: takeEvery, takeLatest, ... đây là những Effect Creator.

### 

### Task là tiến trình chạy dưới background. Trong redux-saga có thể có nhiều tác vụ chạy song song. Bạn tạo các nhiệm vụ bằng cách sử dụng hàm fork

* **Blocking** nghĩa là khi Redux Saga middleware nhận được một **Effect** nó tạm dừng Saga lại chờ cho đến khi câu lệnh đó thực thi xong rồi mới tiếp tục thực thi các **Effect** tiếp theo trong Saga
* **A Non-blocking** nghĩa là **Saga** sẽ được tiếp tục ngay lập tưc sau khi yield một **Effect**
* **Watcher:** Là một **Saga** sẽ theo dõi tất cả những **action** được gửi đến middleware và sẽ thông báo cho **Worker** thực hiện tác vụ tương ứng.
* **Worker:** Là một **Saga** sẽ thực thi các hành động bên trong nó mỗi khi nhận được thông báo từ **Watcher**.
* **saga** phải là một hàm trả về một Generator Object. Middleware đi qua Generator Object va thực thi tất cả yielded Effects
* Trong **redux-saga**, bạn có thể thực hiện các nhiệm vụ (task) nền non-block, bằng 2 hiệu ứng (effect), là **Fork và Spawn**. Vậy làm sao để phân biệt **Fork** và **Spawn** trong redux-saga
  + 1, **Fork** được sử dụng để tạo nhiều task nền, được đính kèm với task cha.
  + 2, **Spawn** cũng giống như fork, nó khác ở chổ là tạo ra cái task nền tách rời với task cha.