Redux: <https://www.youtube.com/channel/UCjgL8tsdzsqOg0hg22VnnwA>

The three most widely used saga commands are:

* take(ACTION\_NAME) — wait for an action of ACTION\_NAME to be dispatched. Returns the action object
* put(action) — dispatch an action to the redux store
* select(selector) — apply a selector to the current state. A selector is simply a function that takes in whole Redux state and returns something
* all – compose all the sagas.

<https://goshacmd.com/lazy-auth-redux-saga-flow/>?

* call` is the command to call other sagas/async functions/promises

======

<https://riad.blog/2015/12/28/redux-nowadays-from-actions-creators-to-sagas/>

Redux suggests using middlewares (especially [thunk middleware](https://github.com/gaearon/redux-thunk" \t "_blank)). Basically the idea is, if you need to trigger side effects, use an action creator: a function that returns a function that can do any async call needed and dispatch whatever action you want.

Using this approach can quickly lead to complex and very difficult to test actions creators. That’s where redux-saga comes in. It defines the concept of a Saga, a declarative and well organized way to express the side-effects (Timeouts, API calls …). So instead of writing an action creator using thunk middleware, we continue dispatching synchronous actions, but instead of having a reducer handling those actions, we will have a Saga taking this action and **yielding** effects (simple javascript objets defining the async actions to perform).

[**https://www.youtube.com/watch?v=SrG99P3D-LA&list=PLBQolri9XOIzbOlPndtIJoIgoKQ1HenTo&index=7**](https://www.youtube.com/watch?v=SrG99P3D-LA&list=PLBQolri9XOIzbOlPndtIJoIgoKQ1HenTo&index=7)

**ReduxSaga:**

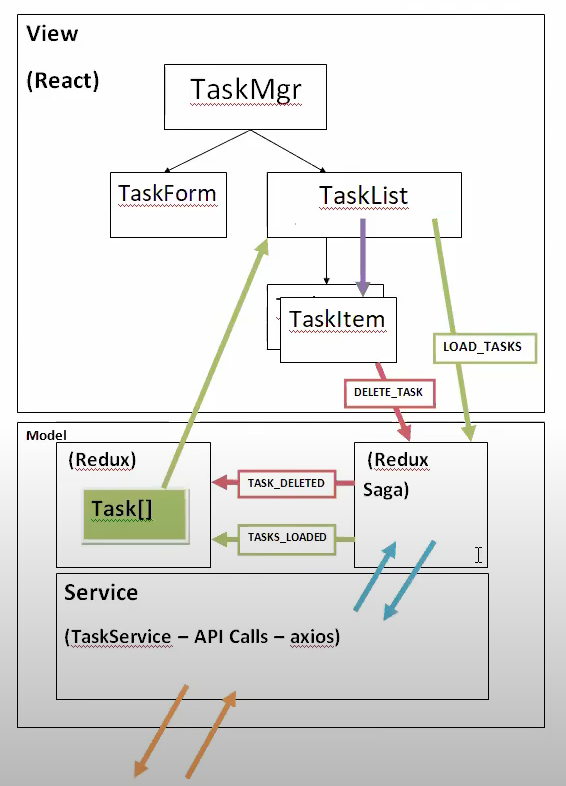
* Is a middleware, every function is a generator function
* Is a library that aims to make application side effects easier to manage.: Side effects are managed easily by the generator functions. The function runs and then waits for the next() to be executed on its instance so that it can run again. This is made possible because of the yield keyword.
* Is capable of listening to the actions being dispatched as well as dispatching actions

**Types of sagas:**

*Watcher Saga*: Watches for the action being dispatched and would assign a worker saga for the action.

*Worker saga:* Would do the work, get the response (from an Api mostly) and would dispatch an action.

*Root Saga:* Composition of all the sagas that we create.

****

Designing:----

St.1 : Make the mind map like this

St.2 : Add All the action types in the actionTypes.js

St.3 : among all the actions except a few actions, all the actions would be listened by the reducer. Some will be only listened by the redux saga and would be then redirected(dispatched) to the redux.