**REDUX SAGA**

Redux Middleware : cùng cấp 1 tầng ở giữa việc dispatch action và nhận action để gửi lên reducer. Thường được dùng trong việc logging, reporting, async api, routing…

* Redux Logger
* Redux Thunk
* Redux Saga

Redux Saga : dùng để quản lí các side-effects

* Đăng kí với Redux Store để sử dụng SagaMiddleware

*import* { createStore, applyMiddleware } *from* 'redux'  
*import* createSagaMiddleware *from* 'redux-saga'  
  
*import* reducer *from* './reducers'  
*import* mySaga *from* './sagas'  
  
// create the saga middleware  
*const* sagaMiddleware = createSagaMiddleware()  
// mount it on the Store  
*const* store = createStore(  
 reducer,  
 applyMiddleware(sagaMiddleware)  
)  
  
// then run the saga  
sagaMiddleware.run(mySaga)  
  
// render the application

* Dispatch action



* Khởi tạo nằm vùng, mỗi lần thấy action xuất hiện thì thực hiện 1 hành động tương ứng (xử lí data xong dispatch action nào đó)

*import* { call, put, takeEvery, takeLatest } *from* 'redux-saga/effects'  
*import* Api *from* '...'  
  
// worker Saga: will be fired on USER\_FETCH\_REQUESTED actions  
*function*\* fetchUser(action) {  
 *try* {  
 *const* user = *yield* call(Api.fetchUser, action.payload.userId);  
 *yield* put({type: "USER\_FETCH\_SUCCEEDED", user: user});  
 } *catch* (e) {  
 *yield* put({type: "USER\_FETCH\_FAILED", message: e.message});  
 }  
}  
  
/\*  
 Starts fetchUser on each dispatched `USER\_FETCH\_REQUESTED` action.  
 Allows concurrent fetches of user.  
\*/  
*function*\* mySaga() {  
 *yield* takeEvery("USER\_FETCH\_REQUESTED", fetchUser);  
}  
  
/\*  
 Alternatively you may use takeLatest.  
  
 Does not allow concurrent fetches of user. If "USER\_FETCH\_REQUESTED" gets  
 dispatched while a fetch is already pending, that pending fetch is cancelled  
 and only the latest one will be run.  
\*/  
*function*\* mySaga() {  
 *yield* takeLatest("USER\_FETCH\_REQUESTED", fetchUser);  
}  
  
*export* *default* mySaga;

Các khái niệm :

* Effect ➣ JS Obj chứa thông chỉ thị hành động cho saga middleware

VD: call(myFunc, ‘arg1’, ‘arg2’) ➣ saga sẽ chạy myFucn(‘arg1’, ‘arg2’) và trả về kq cho đối tượng đã thực hiện effect

* Task ➣ process chạy ngầm

*import* {fork} *from* "redux-saga/effects"  
  
*function*\* saga() {  
 ...  
 *const* task = *yield* fork(otherSaga, ...args)  
 ...  
}

* Blocking / Non-blocking

Blocking: bắt buộc thực hiện xong

Non-blocking: chỉ trigger thực hiện

<https://redux-saga.js.org/docs/api/#blocking--non-blocking>

*import* {call, cancel, join, take, put} *from* "redux-saga/effects"  
  
*function*\* saga() {  
 *yield* take(ACTION) // Blocking: will wait for the action  
 *yield* call(ApiFn, ...args) // Blocking: will wait for ApiFn (If ApiFn returns a Promise)  
 *yield* call(otherSaga, ...args) // Blocking: will wait for otherSaga to terminate  
  
 *yield* put(...) // Non-Blocking: will dispatch within internal scheduler  
  
 *const* task = *yield* fork(otherSaga, ...args) // Non-blocking: will not wait for otherSaga  
 *yield* cancel(task) // Non-blocking: will resume immediately  
 // or  
 *yield* join(task) // Blocking: will wait for the task to terminate  
}

* Watcher / Worker

Watcher: theo dõi action, fork a worker on every action

Worker: handle action and terminate

*function*\* watcher() {  
 *while* (true) {  
 *const* action = *yield* take(ACTION)  
 *yield* fork(worker, action.payload)  
 }  
}  
  
*function*\* worker(payload) {  
 // ... do some stuff  
}

API :

* createSagaMiddleware

import createSagaMiddleware from 'redux-saga'

import reducer from './path/to/reducer'

export default *function* configureStore(*initialState*) {

**// Note: passing middleware as the last argument to createStore requires redux@>=3.1.0**

*const* sagaMiddleware = createSagaMiddleware()

  return {

    ...createStore(reducer, initialState, applyMiddleware(**/\* other middleware, \*/**sagaMiddleware)),

    runSaga: sagaMiddleware.run

  }

}

* middleware.run(saga, ...args)

*const* store = configureStore()

store.runSaga(rootSaga) // contain child saga

Set up store trước rồi mới run saga

**Generator**

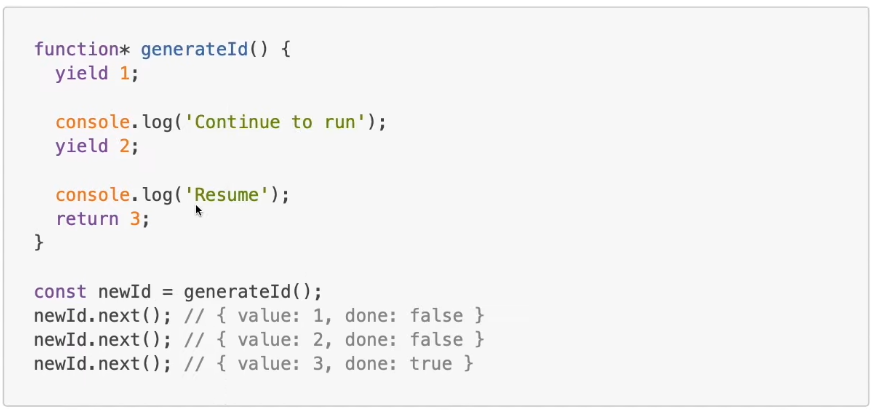
**1. Javascript Generator**

Obj trả về từ generator Func()

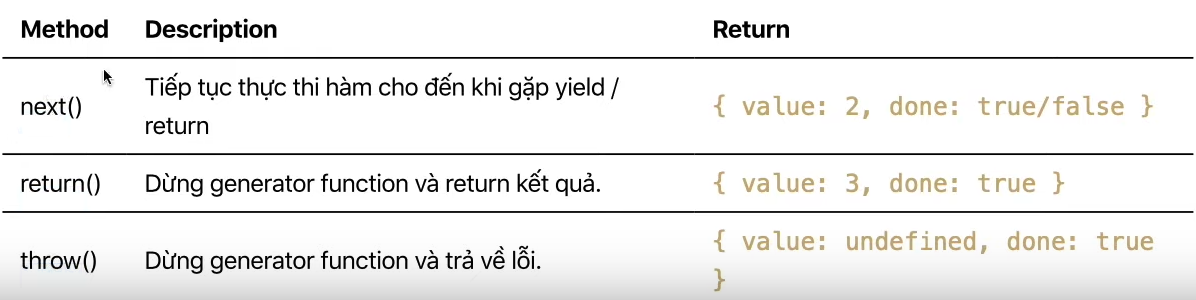
K thể gọi trực tiếp, chỉ return từ gene Func

Cú pháp gene Func() :

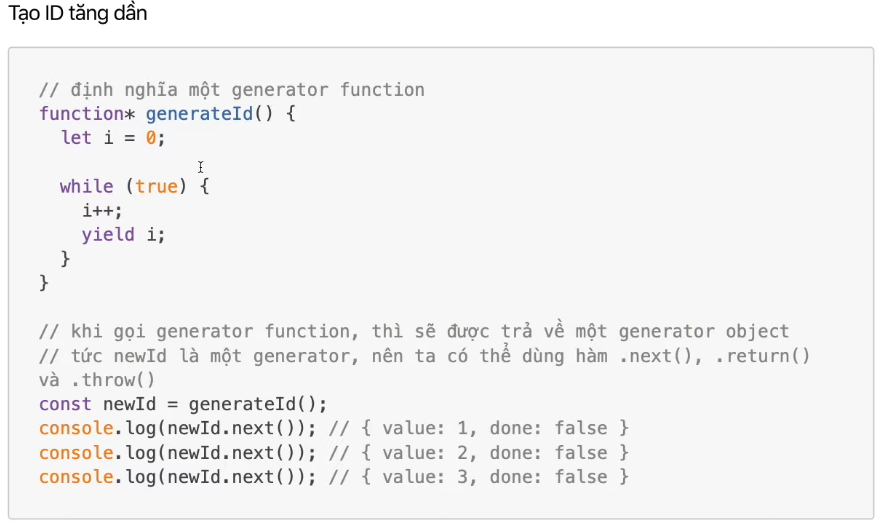
* gene Func() có thể dừng giữa chừng và chạy tiếp sau
* Mỗi lần gọi, hàm chạy cho tới khi yield or return



generator newId có 3 hàm :



**2. Ví dụ**







**Saga**

**1. Header**

<a  
 className="header\_\_link header\_\_title"  
 href="https://youtube.com/easyfrontend"  
 target="\_blank"  
 rel="noopener noreferrer"   
>  
 Easy Frontend  
</a>

rel bắt buộc khi target blank

<NavLink  
 exact  
 className="header\_\_link"  
 to="/photos"  
 activeClassName="header\_\_link--active"  
>  
 Redux Project  
</NavLink>

active là link khi đã click

**2. Font**

add link gg Font vào index.html

add font-family vào index.css

**3. Xử lý relative imports với jsconfig.json**

**4. Formik**

**## Custom Field**

**Cầu nối giữa UI control và Formik**

**UI control là 1 controlled component với props:**

- 4 thứ quan trọng trong 1 control

1. name: tên xác định control
2. value: giá trị của control
3. onChange: trigger hàm này với giá trị mới khi có thay đổi
4. onBlur: xác định khi nào thì control này bị touched

- Form auto bind reset, submit

- Field độc lập, k phụ thuộc => FastField, k re-render khi các field khác thay đổi

1. name
2. component={InputField}

* label
* placeholder

lưu ý – initialValues --

- Field liên quan => Field, re-render khi các field có liên quan thay đổi

**## Random Photo control**

**RandomPhoto => DOM, UI Component**

**Props**

* name
* imageUrl
* onImageUrlChange
* onRandomButtonBlur

**RandomPhotoField => Cầu nối với Formik**

**Formik => Lưu dữ liệu**

**5. Yup để validate**

Custom field k có className nên phải định nghĩa

**Redux Toolkit**