**React saga**

1. **Material ui, material icon.**

asd

1. **ESLint**

**Cấu hình:**

* .eslintrc.js:

Module.export = {

Parser: “babel-eslint”,

Rules: {  
 semi: 1 // hiển thị lỗi, code vân chạy.

Quotes: [2, ‘ single ‘ ] // quy định dấy nháy.

‘react/prop-types’: 1.

‘react/jsx-max-props-per-line’: 1 // 1 props 1 dòng

}

npm install eslint-plugin-react --save-dev

**Sử dụng một số plugin chứa eslint sắn, ta chỉ việc extends lại:**

npm install --save-dev eslint-config-airbnb

1. **Prettier**

**Hỗ trợ format code, nhiều ngôn ngữ.**

**Kết hợp với ESLint & perttier:**

* Perttier format code dựa trên rules của ESLint.
* ESLint show errors của eslint và pretties.

Cài đặt plugin Prattier trong VS code

yarn add --dev eslint-config-prettier eslint-plugin-prettier prettier

**Cấu hình check lỗi và sửa lỗi toàn bộ.**

"linting": "eslint src",

    "prettier": "prettier \"src/\*\*/\*.{js,css}\"",

    "format": "npm run prettier -- --write"

* Env: khai báo biến môi trường.

1. **Husky and lint-staged : Package.json**

  "husky": {

    "hooks": {

      "pre-commit": "lint-staged"

    }

  },

  "lint-staged": {

    "\*\*/\*.{js,css}": [

      "eslint src",

      "npm run format",

      "git add"

    ]

  }

**5.Editor config**

Thống nhất cấu hình code.

Add plugin **editorconfig.**

**Search: editorconfig react.**

1. **File:**

**File redux: file cấu hình store, tích hợp các middleware.**

**Install redux react-dedux redux-thunk**

**Tích hợp redux-thunk:** tích hợp middleware vào trong redux-thunk , sử lí sideEffect.

**File actions :** gọi API, despatch các action.

**Tạo class axiosService:**

1. **Axios:**

Tìm mục interceptors.

1. **Thư viện hiển thị thông báo: react-toastify.**

npm install --save react-toastify

Object detrutoring: gán thì phải đóng ngoặc.

**Những error** liên quan đến **Axios** chúng sẽ sử lý cho chúng ta. Trả về **data error.**

1. **Redux:**

Cài đặt cấu hình redux.

**Tạo root reducer :**

**-** import**: combineReducer** from **Redux**.

**Tạo thư mục redux :** Tạo file cấu hình **store.** Tích hợp các **middleware.**

**-** import: { **createStore, compose, applyMiddleware }** from **Redux.**

**-** import **thunk** from **redux-thunk**.

**-** Tích hợp **Redux dev tool.**

**-** Tạo **function** chứa: biến quản lý **store, middleware = [ thunk ]** chứa các **middleware**. **Enhancers = [ applyMiddleware(...middleware) ]**

1. **Tích hợp Axios và xây dưng Base service**

Tạo **class** **axiosService** : Tái sử dụng **code**. Mở rộng

- **import** vô sử dụng 🡪 nó sẽ gọi toán tử **New** tạo ra **1** **instance** thuộc về **class** **axiosService** . Đồng thời nó chạy **constructor** đầu tiên. Biến **this** sẽ gọi trong **class.**

Từ khóa trong thư viện **axios : interceptors** 🡪 trả về **response** và **error**.

// Add a response interceptor

axios.interceptors.response.use(function (response) {

// Do something with response data

return response;

}, function (error) {

// Do something with response error

return Promise.reject(error);

});

**Tạo thư mục apis :** Chứa các hàm gọi **API** riêng biệt cho từng **module**. Từng **module** sẽ sử dụng **axiosService** gọi **API**

1. **Viết action type trong Constant**

**Trong mục constant :**

**-** Tạo các biến **const.**

**Trong mục action :**

**-** Gọi **API** dispatch các **action.**

1. **Dispatch action ra sử dung**

**Trong mục container :**

Import **{ connect }** from **‘ react-redux ’:**

**Import**

1. **Redux-saga: Generator function.**

Là các function, các quy trình nền, chương trình con, có khả năng tạm dừng và tiếp tục chạy. Có khả năng thoát ra khỏi hàm sau đó nhập lại vào trong hàm chạy lại. Và mỗi lần thoát ra vào hàm như vậy ta có thể lưu trữ các biến.

Cú pháp: dấu \* có thể xuất hiện bất kỳ điểm nào giữa từ khóa funtion và tên hàm.

Từ khóa **Yield**: chỉ danh riêng cho **generator function.**

+ Trả về giá trị từ **generator function**. Tương đồng với với tứ khóa **return**.

+ Trì hoãn việc thực thi các đoạn code tiếp theo.

+ Dừng lại. Sau đó bắt đầu từ điểm kết thúc.

**Generator funtion :** trả về các **iterators**

+ *Là một chuỗi các đối tượng ( giá trị ),* hỗ trợ lấy giá trị tiếp theo ( **yield** tiếp theo ) trong chuỗi bằng hàm **next**.

console.log(execute.next());

{value: 2019, done: false} : false là còn giá trị.

{value: "Hello Generator", done: true} : true không còn giá trị.

**Phải gán vào biến.**

**Kết thúc**

**+** **Return**: kết thúc luôn **generator function.**

**+ Throw error** : kết thúc luôn **generator function.**

**Vòng lặp vô tận:** rất hữu ích trong **generator function.**

**+** Hàm **chỉ chạy** mỗi khi ứng dụng của chúng ta gọi **Next().**

**Generator trong generator:**

**+ Yield\* funName() :** Sẽ được nhường quyền cho một **generator** function khác.

**Tại sao sử dụng generator:**

Sinh ra để giải quyết luồng điều khiển không đồng bộ **thành đồng bộ** và code sẽ rõ ràng **dễ** **đọc** hơn.

1. **Cài đặt Redux-saga:**

+ Cài đặt package **redux-saga.**

+ Kết nối **saga** **Middleware** với **store**.

+ Lưu ý khi **applyMiddware** có phân biệt thứ tự.

+ Hàm **run** để chương trình con bắt đầu **theo dõi các action.**

**Root saga:**

**B1:** import

Giống như **combineReducers: rootReducer :** điểm bắt đầu của **reducer**.

**Root saga** ( **điểm bắt đầu** ): là **entry point** của **saga ( generator function )**. Nơi đọc đầu tiên, điều phối các **saga** khác.

Khởi động tất cả các **sagas** khác để chạy nền **( background )**, theo dõi và kích hoạt xử lý các **action.**

Có thể sử dụng đồng thời cả **thunk** và **saga**: vì môi **saga** sẽ lắng nghe, theo dõi, phản hồi các **action** cụ thể.

**Saga** được gọi khi đăng kí theo dõi các **action**.

1. **Thêm, sửa, xóa bằng form.**

**Thẻ form :** bao bọc lại.

Tích hợp **form** vào **Redux.**

+ Viết **action** hiển thị nội dung bên trong **model: showModal, hideModal, changeModalTitle, changModalContent**( component )

+ Tạo **reducer 🡺** sử dụng ở **container.**

Tạo **component modal connect** tới **store**

1. **Redux – form**

Giúp quản lý **state form** trên **redux**. Hỗ trợ mạnh **validation**

1. **Api :**

**+ ReduxForm( ) :** connect tới **Store** của **redux-from.**

**+ Props :** sự kiện thao tác với ô input xem có validation

**+ Field :** input, ...

**+ FormSelection :** Gom nhóm lại.

**+ FormValues :** Lấy được values của form.

**+ FormValuesSelector :** Lấy được value của form , các trạng thái của **form.**

**+ Reducer :** của **form.**

**+ SubmissionError :** form không hợp lệ thì sẽ không được quyền **submit .**

**+ Action Creators :** action tương ứng ví dụ, **onchange**, **onblur.**

1. **Đăng kí form-reducer vào project**

**Thực hiện 4 bước cài đặt.**

**Submit Form:**

 const { handleSubmit } = props;

  <form onSubmit={handleSubmit(handlerSubmitForm)}> <form/>

**Sử dụng matiral-ui in redux-form**

**Tạo component :** sài chung cho **material-ui.**

**+** Chia theo component.

const renderTextField = ({

  label,

  input, // có sự kiện của input mà trước đây html on..

  meta: { touched, invalid, error }, // kiểm tra xem hợp lệ hay chưa..

  ...custom

}) => (),

1. **redux-form Validation**

**Synchronous Validation :** ở phía **client** .

**+ Cách 1 :** **Field – Level validation** cho từng field.

Sử dụng biến chứa hàm **validate = { }**

+ **Cách 2:** viết hàm hiển thị lỗi chung.

Không hợp lệ thì không cho **click submit 🡺 API Props. : State** của **component** chuyển thành **props**.: **invalid, submitting**

1. **Thêm dữ liệu**

**Viêt phương thức post** : nhận vào **url, body.**

1. **Nâng cấp tìm kiếm**

**Gọi Api :** Trong **Json-server :** từ khóa **search**

Thêm **param** trong **action**.

**Install query-string**

1. **Edit Task**

**Truyền task 🡺 taskboard**

**Tạo 1 state task** trong reducer là **taskEditing**.

Click thêm mới thì **taskEditing : null.**

Click vào sửa -> mở **form**. Truyền nội dung vào cách 1. Cách 2 gọi **state** trong **redux**

Đổ dữ liệu vào **value**: trong **redux-form -> initialValues :** Khởi tạo giá trị cho form**.** Vào trong **mapStateToProps**.

**2) Cập nhật dữ liệu lên server**

**Viết 3 action :**

**Thêm field :** hiển thị status. Chỉ sửa mới hiển thị.

**3) Viết hàm update API**

**Tham số là Data va id**

**Ctrl + k + 0 :** đóng dòng lệnh.

const { data, status: statusCode } = resp;

: cách thay đổi tên gọi.

1. **Xóa công việc**

**Tạo giao diện:**

**Tạo ra action, reducer tương ứng :**

**Viết hàm delete API truyền vào id :**.

**deleteSuccess( id ) :** Không trả về **data** vừa xóa. **Data** nhận được là rỗng

1. **sadas**