Tài liệu web

[1. Cấu trúc project 2](#_Toc15544093)

[A. Cấu trúc 2](#_Toc15544094)

[B. Yêu cầu & Cách chạy 3](#_Toc15544095)

[C. Mô tả chi tiết 3](#_Toc15544096)

[2. Checklist khi viết code 3](#_Toc15544097)

[3. Quy tắc viết css 4](#_Toc15544098)

[4. CÁCH VIẾT COMPONENT 5](#_Toc15544099)

[A. Những luật cơ bản 5](#_Toc15544100)

[B. So sánh class vs React.createClass vs stateless 6](#_Toc15544101)

[C. Đặt tên 7](#_Toc15544102)

[D. Khai báo 8](#_Toc15544103)

[E. Căn chỉnh mã nguồn 9](#_Toc15544104)

[F. Dấu nháy đơn và nháy kép 9](#_Toc15544105)

[G. Khoảng trắng 9](#_Toc15544106)

[H. Props 10](#_Toc15544107)

[I. Refs 14](#_Toc15544108)

[J. Dấu ngoặc đơn 14](#_Toc15544109)

[K. Thẻ 14](#_Toc15544110)

[L. Phương thức 15](#_Toc15544111)

[M. Cách sắp xếp hàm 17](#_Toc15544112)

[5. Ví dụ 18](#_Toc15544113)

[a) Lấy dữ liệu ở server: 19](#_Toc15544114)

[b) Lấy dữ liệu ở client: 20](#_Toc15544115)

[c) Thay đổi dữ liệu: 22](#_Toc15544116)

# Cấu trúc project

## Cấu trúc

├── componentDashs/ # Các thành phần dùng chung của dashBoard

│ └── Table

│

├── componentWebs/ # Các thành phần dùng chung của web

│ └── Home

│

├── config/ # cấu hình dùng chung router web

│

├── controllers/ # Các định nghĩ lấy/thêm dữ liệu cho view

│ └── components/ # Các định nghĩ lấy/thêm dữ liệu được viết ở │. │ folder components

│ │

│ └── layouts/ # Các định nghĩ lấy/thêm dữ liệu được viết ở │ │ folder layouts

│ │

│ └── pages/ # Các định nghĩ lấy/thêm dữ liệu được viết ở │ folder pages

│

├── layout/ # cấu hình dùng chung (HOC template) cho giao │ diện

│

├── models/ # định nghĩa các models cho redux

│

├── page/ # Các trang chính trong nextjs

│ └── \_documents.js

│ │

│ └── \_app.js

│ │

│ └── index.js # file view của web

│ │

│ └── abouts/ # folder view của web

│

├── services/ # Các định nghĩa gọi API

│ └── advertismentServices.js

│

├── static/ # Chứa các file tĩnh css, images,js,…

│ │

│ └── dashBoard # file less của componentDashs

│ │

│ └── public # file css, images, js, fonts chung của web

│ │

│ └── web # file css, images, js của web

│

├── templates/ # Chứa các folder mẫu

│

├── next.config.js

├── package.json # Là file cấu hình, chứa thông tin, phiên bản của các gói, thư viện cần │ thiết để project hoạt động bình thường.

├── .env

├── .env.production

## Yêu cầu & Cách chạy

Yêu cầu:

node > 8

npm or yarn

Cài modules:

npm install or yarn install

Chạy:

npm run dev or yarn dev

## Mô tả chi tiết

### config: cấu hình chung (router web)

### layouts: cấu hình chung(HOC template) cho giao diện

### node\_modules: các thư viện node

### components: các thành phần dùng chung

### pages: các trang chính trong nextjs, chỉ chứa các trang view của web

quy tắc đặt tên: product.js (product.js)

### model: các định nghĩa model cho redux

quy tắc đặt tên: tenModel.js (productModel.js)

### services: các định nghĩa gọi API

quy tắc đặt tên: tenService.js (productService.js)

### static: chứa các thành phần tĩnh như css, images, fonts...

### utils: các công cụ dùng chung

### controllers: các định nghĩa phương thức lấy/thêm dữ liệu cho view

quy tắc đặt tên: tenController.js (productController.js)

# Checklist khi viết code

|  |
| --- |
| **Layout màn hình Backend** |
| Check đầy đủ các item |
| Check đúng các Component type của các item |
| Thống nhất font chữ: 14px, font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'PingFang SC', 'Hiragino Sans GB', 'Microsoft YaHei', 'Helvetica Neue', Helvetica, Arial, sans-serif, 'Apple Color Emoji', 'Segoe UI Emoji', 'Segoe UI Symbol' |
| Kích thước các textbox, text area phù hợp với content |
| Check giá trị pulldown |
| **Layout màn hình (Font End)** |
| Check đúng với screen design của khách hàng |
| Check responsive với các màn hình mobile , destop, Ipad |
| **Event trên màn hình** |
| Check các button |
| Check các msg thông báo trên màn hình : Ngôn ngữ, chính tả, đồng nhất,… |
| Check các link router có hoạt động không |
| **Validate** |
| Item là unique: đã check trùng lặp chưa ? |
| Đã thực hiện check item require chưa ? |
| Đã thực hiện check maxlength chưa ? |
| Đối với item kiểu dữ liệu int: chỉ cho phép nhập số ? |
| Đối với kiểu string, đã xử lý cho các ký tự thẻ html khi addnew, edit chưa? |
| Đối với định dạng ảnh,đã cho chọn đúng định dạng ảnh chưa (jpg,png,..). Giới hạn dung lượng ảnh? |
| Đối với kiểu date, đã sử dụng date picker chưa ? |
| **Search** |
| Với trường hợp search, đã cắt khoảng trắng ở 2 đầu từ khóa tìm kiếm chưa? |
| Đối với item tìm kiếm chính xác: đã thực hiện search chính xác chưa ? |
| Đối với item không phải là tìm kiếm chính xác: đã search được tương đối chưa |
| **Sort (nếu có)** |
| Check các item cần sort :Đúng và đủ theo spec |
| Check chức năng sort hoạt động không |
| **Upload file** |
| Check dung lượng file upload |
| Check định dạng file đúng với design |

# Quy tắc viết css

Theo quy ước đặt tên BEM

Block ––– Khu vực của đối tượng cần viết CSS.

Element – Các phần tử bên trong của Block.

Modifier – Tính chất của phần tử.

Ý nghĩa cách đặt tên:

Class đó được sử dụng ở đâu.

Class đó có nhiệm vụ gì.

Quan hệ với các Class thế nào.

Chi tiết:

// Một Block(khối) độc lập có thể là header, nav, article, footer, div...

.stick-man {}

// Element head,arm,feet (Block con) phụ thuộc vào Block stick-man

.stick-man\_\_head {}

.stick-man\_\_arm {}

.stick-man\_\_feet {}

// Tính chất red, blue của Block stick-man

.stick-man--red {}

.stick-man--blue {}

// Tính chất blue, yellow của Block stick-man\_\_head

.stick-man\_\_head--blue {}

.stick-man\_\_head--yellow {}

==> Ý nghĩa các dấu phân cách:

- 1 gạch ngang : Ngăn cách tên

 \_\_ : 2 gạch dưới: Đặt trước block con

-- : 2 gạch ngang : Đặt trước thuộc tính

# CÁCH VIẾT COMPONENT

Tuân theo chuẩn Airbnb: [*https://github.com/UETCodeCamp/jsx-style-guide*](https://github.com/UETCodeCamp/jsx-style-guide)

## Những luật cơ bản

* Chỉ chứa một React Component trong 1 file.
* Tuy nhiên, những component có khả năng sử dụng lại([Stateless Component, hoặc Pure Components](https://facebook.github.io/react/docs/reusable-components.html#stateless-functions)) có thể viết chung một file. eslint: [react/no-multi-comp](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/no-multi-comp.md#ignorestateless).
* Luôn luôn sử dụng cú pháp JSX.  - Không sử dụng React.createElement chung với cú pháp JSX.

## So sánh class vs React.createClass vs stateless

* Nếu Component có state hoặc refs, nên sử dụng class extends React.Component thay vì React.createClass. eslint: [react/prefer-es6-class](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/prefer-es6-class.md) [react/prefer-stateless-function](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/prefer-stateless-function.md)
* // tệ
* const Listing = React.createClass({
* // ...
* render() {
* return <div>{this.state.hello}</div>;
* }
* });
* // tốt
* class Listing extends React.Component {
* // ...
* render() {
* return <div>{this.state.hello}</div>;
* }

}

Và nếu trong Component không có state hoặc refs, nên sử dụng khai báo hàm (không phải arrow function) thay vì class:

// tệ

class Listing extends React.Component {

render() {

return <div>{this.props.hello}</div>;

}

}

// tệ (dựa vào tên hàm để suy luận thì rất đau đầu)

const Listing = ({ hello }) => (

<div>{hello}</div>

);

// tốt

function Listing({ hello }) {

return <div>{hello}</div>;

}

## Đặt tên

* **Phần mở rộng(extensions)**: Sử dụng phần mở rộng .js hoặc.jsx cho React Components.
* **Tên file**: Sử dụng chuẩn PascalCase cho tên file. Ví dụ: ReservationCard.jsx.
* **Tên tham chiếu(Reference Naming)**: Sử dụng PascalCase cho React components và dùng camelCase cho các đối tượng(instances) của chúng. eslint: [react/jsx-pascal-case](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-pascal-case.md)
* // tệ
* import reservationCard from './ReservationCard';
* // tốt
* import ReservationCard from './ReservationCard';
* // tệ
* const ReservationItem = <ReservationCard />;
* // tốt

const reservationItem = <ReservationCard />;

* **Đặt tên Component**: Sử dụng tên file trùng với tên component. Ví dụ: ReservationCard.jsx nên có tên tham chiếu là ReservationCard. Tuy nhiên, đối với các component gốc của một thư mục, hãy sử dụng index.jsx làm tên file và sử dụng tên thư mục làm tên component:
* // tệ
* import Footer from './Footer/Footer';
* // tệ
* import Footer from './Footer/index';
* // tốt

import Footer from './Footer';

* **Đặt tên Higher-order Component**: Sử dụng sự kết hợp của Higher-order component và tên của component đuợc truyền như displayName(tên hiển thị) trên component đuợc tạo ra. Ví dụ component bậc cao withFoo(), khi truyền một component Bar sẽ tạo ra một component với displayName của withFoo(Bar).

Tại sao? Vì displayName của component có thể đuợc sử dụng bởi những công cụ phát triển hoặc trong các thông báo lỗi, và có một giá trị mà thể hiện rõ mối quan hệ này sẽ giúp chúng hiểu rõ chuyện gì đang xảy ra.

// tệ

export default function withFoo(WrappedComponent) {

return function WithFoo(props) {

return <WrappedComponent {...props} foo />;

}

}

// tốt

export default function withFoo(WrappedComponent) {

function WithFoo(props) {

return <WrappedComponent {...props} foo />;

}

const wrappedComponentName = WrappedComponent.displayName

|| WrappedComponent.name

|| 'Component';

WithFoo.displayName = `withFoo(${wrappedComponentName})`;

return WithFoo;

}

* **Đặt tên Props**: Tránh sử dụng tên props của DOM Component cho mục đích khác.

Tại sao? Mọi nguời mong đợi props như style và className có ý nghĩa riêng. Việc thay đổi mục đích sử dụng của API gốc làm cho mã khó đọc và khó bảo trì hơn, thậm chí có thể gây ra lỗi.

// tệ

<MyComponent style="fancy" />

// tệ

<MyComponent className="fancy" />

// tốt

<MyComponent variant="fancy" />

## Khai báo

* Không nên sử dụng displayName để đặt tên cho các Components. Thay vào đó, đặt tên cho các Components bằng references(tham chiếu).
* // tệ
* export default React.createClass({
* displayName: 'ReservationCard',
* // một số thứ khác
* });
* // tốt
* export default class ReservationCard extends React.Component {

}

## Căn chỉnh mã nguồn

* Căn chỉnh cho cú pháp JSX. eslint: [react/jsx-closing-bracket-location](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-closing-bracket-location.md) [react/jsx-closing-tag-location](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-closing-tag-location.md)
* // tệ
* <Foo superLongParam="bar"
* anotherSuperLongParam="baz" />
* // tốt
* <Foo
* superLongParam="bar"
* anotherSuperLongParam="baz"
* />
* // Nếu props phù hợp trong một dòng thì giữ nó trên cùng một dòng
* <Foo bar="bar" />
* // Component con được thụt lề bình thường
* <Foo
* superLongParam="bar"
* anotherSuperLongParam="baz"
* >
* <Quux />

</Foo>

## Dấu nháy đơn và nháy kép

* Luôn luôn sử dụng dấu ngoặc kép (") cho các thuộc tính JSX, nhưng dấu nháy đơn (') cho tất cả các JS khác. Eslint: [jsx-quotes](https://eslint.org/docs/rules/jsx-quotes)

Tại sao? Vì các thuộc tính HTML thông thường thường sử dụng dấu ngoặc kép thay vì đơn, vì vậy thuộc tính JSX cũng như thế.

// tệ

<Foo bar='bar' />

// tốt

<Foo bar="bar" />

// tệ

<Foo style={{ left: "20px" }} />

// tốt

<Foo style={{ left: '20px' }} />

## Khoảng trắng

* Luôn luôn có duy nhất một kí tự space(khoảng trắng) trong thẻ tự đóng. eslint: [no-multi-spaces](https://eslint.org/docs/rules/no-multi-spaces), [react/jsx-tag-spacing](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-tag-spacing.md)
* // tệ
* <Foo/>
* // rất tệ
* <Foo />
* // tệ
* <Foo
* />
* // tốt

<Foo />

* Không dùng khoảng trắng giữa giá trị bên trong ngoặc nhọn. eslint: [react/jsx-curly-spacing](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-curly-spacing.md)
* // tệ
* <Foo bar={ baz } />
* // tốt

<Foo bar={baz} />

## Props

* Luôn luôn sử dụng camelCase khi đặt tên prop (camelCase : viết hoa chữa cái đầu của các từ , từ đầu tiên của cụm thì viết thường)
* // tệ
* <Foo
* UserName="hello"
* phone\_number={12345678}
* />
* // tốt
* <Foo
* userName="hello"
* phoneNumber={12345678}

/>

* Bỏ giá trị của prop khi nó thực sự rõ ràng là true. eslint: [react/jsx-boolean-value](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-boolean-value.md)
* // tệ
* <Foo
* hidden={true}
* />
* // tốt
* <Foo
* hidden
* />
* // tốt

<Foo hidden />

* Luôn luôn sử dụng prop alt trong thẻ <img>. Nếu giá trị của thẻ là NULL , alt có thể là một chuỗi rỗng hoặc <img> phải có thuộc tính role="presentation". eslint: [jsx-a11y/alt-text](https://github.com/evcohen/eslint-plugin-jsx-a11y/blob/master/docs/rules/alt-text.md)
* // tệ
* <img src="hello.jpg" />
* // tốt
* <img src="hello.jpg" alt="Me waving hello" />
* // tốt
* <img src="hello.jpg" alt="" />
* // tốt

<img src="hello.jpg" role="presentation" />

* Không dùng các từ "image", "photo", hoặc "picture" trong <img> alt props. eslint: [jsx-a11y/img-redundant-alt](https://github.com/evcohen/eslint-plugin-jsx-a11y/blob/master/docs/rules/img-redundant-alt.md)

Tại sao? Screenreaders đã tự hiểu img elements là image(ảnh), vì vậy không cần khai báo thông tin này trong alt

// tệ

<img src="hello.jpg" alt="Picture of me waving hello" />

// tốt

<img src="hello.jpg" alt="Me waving hello" />

* Chỉ sử dụng [ARIA roles](https://www.w3.org/TR/wai-aria/roles#role_definitions). eslint: [jsx-a11y/aria-role](https://github.com/evcohen/eslint-plugin-jsx-a11y/blob/master/docs/rules/aria-role.md) hợp lệ, và không trừu tượng. [jsx-a11y/aria-role](https://github.com/evcohen/eslint-plugin-jsx-a11y/blob/master/docs/rules/aria-role.md)
* // tệ - không phải ARIA roles
* <div role="datepicker" />
* //tệ- ARIA roles trừu tượng
* <div role="range" />
* // tốt

<div role="button" />

* Không dùng accessKey trong các elements. eslint: [jsx-a11y/no-access-key](https://github.com/evcohen/eslint-plugin-jsx-a11y/blob/master/docs/rules/no-access-key.md)

Tại sao ? Sự mâu thuẫn giữa phím tắt và các lệnh bàn phím được những người dùng screenreaders sử dụng làm phức tạp hóa khả năng tiếp cận.

// tệ

<div accessKey="h" />

// tốt

<div />

* Tránh dùng chỉ số của mảng(index) cho thuộc tính key, nên sử dụng một unique ID(định danh duy nhất). ([why?](https://medium.com/@robinpokorny/index-as-a-key-is-an-anti-pattern-e0349aece318))
* // tệ
* {todos.map((todo, index) =>
* <Todo
* {...todo}
* key={index}
* />
* )}
* // tốt
* {todos.map(todo => (
* <Todo
* {...todo}
* key={todo.id}
* />

))}

* Luôn xác định rõ ràng các defaultProp(thuộc tính mặc định) cho tất cả non-required props(thuộc tính không bắt buộc).

Tại sao? propTypes được coi như tài liệu, và cung cấp defaultProps , nghĩa là người đọc mã nguồn của bạn không cần phải đoán quá nhiều. Ngoài ra, nó có thể bỏ qua một số kiểm tra kiểu(type checking).

// tệ

function SFC({ foo, bar, children }) {

return <div>{foo}{bar}{children}</div>;

}

SFC.propTypes = {

foo: PropTypes.number.isRequired,

bar: PropTypes.string,

children: PropTypes.node,

};

// tốt

function SFC({ foo, bar, children }) {

return <div>{foo}{bar}{children}</div>;

}

SFC.propTypes = {

foo: PropTypes.number.isRequired,

bar: PropTypes.string,

children: PropTypes.node,

};

SFC.defaultProps = {

bar: '',

children: null,

};

* Hạn chế lạm dụng toán tử spread cho việc truyền props

Tại sao? Vì bạn có khả năng truyền props không cần thiết xuống Components. Và với React v15.6.1 trờ lên, bạn cần [chuyển các thuộc tính không hợp lệ của HTML sang DOM](https://reactjs.org/blog/2017/09/08/dom-attributes-in-react-16.html).

Ngoại lệ:

* HOCs có thể truyền thẳng props xuống và khai báo propTypes
* function HOC(WrappedComponent) {
* return class Proxy extends React.Component {
* Proxy.propTypes = {
* text: PropTypes.string,
* isLoading: PropTypes.bool
* };
* render() {
* return <WrappedComponent {...this.props} />
* }
* }

}

* Sử dụng toán tử spread đối với prop được khai báo rõ ràng. Điều này có thể đặc biệt hữu ích khi test các React component với cấu trúc beforeEach của Mocha.
* export default function Foo {
* const props = {
* text: '',
* isPublished: false
* }
* return (<div {...props} />);

}

* Ghi chú: Nên lọc các props không cần thiết khi có thể. Ngoài ra, sử dụng [prop-types-exact](https://www.npmjs.com/package/prop-types-exact) để giúp ngăn ngừa lỗi.
* // tốt
* render() {
* const { irrelevantProp, ...relevantProps } = this.props;
* return <WrappedComponent {...relevantProps} />
* }
* // tệ
* render() {
* const { irrelevantProp, ...relevantProps } = this.props;
* return <WrappedComponent {...this.props} />

}

## Refs

* Luôn sử dụng hàm gọi lại(callback) cho khai báo ref. eslint: [react/no-string-refs](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/no-string-refs.md)
* // tệ
* <Foo
* ref="myRef"
* />
* // tốt
* <Foo
* ref={(ref) => { this.myRef = ref; }}

/>

## Dấu ngoặc đơn

* Đóng gói các thẻ JSX trong ngoặc đơn khi chúng kéo dài nhiều dòng. eslint: [react/jsx-wrap-multilines](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-wrap-multilines.md)
* // tệ
* render() {
* return <MyComponent variant="long body" foo="bar">
* <MyChild />
* </MyComponent>;
* }
* // Tốt
* render() {
* return (
* <MyComponent variant="long body" foo="bar">
* <MyChild />
* </MyComponent>
* );
* }
* // Tốt, Khi chỉ có 1 dòng
* render() {
* const body = <div>hello</div>;
* return <MyComponent>{body}</MyComponent>;

}

## Thẻ

* Luôn luôn tự đóng các thẻ(tags) không có con. eslint: [react/self-closing-comp](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/self-closing-comp.md)
* // tệ
* <Foo variant="stuff"></Foo>
* // tốt

<Foo variant="stuff" />

* Nếu Component của bạn có thuộc tính nhiều dòng, hãy đóng thẻ đó trên 1 dòng mới. eslint: [react/jsx-closing-bracket-location](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-closing-bracket-location.md)
* // tệ
* <Foo
* bar="bar"
* baz="baz" />
* // tốt
* <Foo
* bar="bar"
* baz="baz"

/>

## Phương thức

* Sử dụng arrow function để bao đóng các biến cục bộ.
* function ItemList(props) {
* return (
* <ul>
* {props.items.map((item, index) => (
* <Item
* key={item.key}
* onClick={() => làmGìĐó(item.name, index)}
* />
* ))}
* </ul>
* );

}

* Các hàm binding được gọi trong lúc render nên đặt ở trong hàm khởi tạo(constructor). eslint: [react/jsx-no-bind](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/jsx-no-bind.md)

Tại sao? Vì nếu bind trong hàm render thì mỗi khi render, hàm đó lại được tạo mới một lần khiến cho hiệu suất xử lí giảm.

// tệ

class MyComponent extends React.Component {

onClickDiv() {

// làm việc gì đó

}

render() {

return <div onClick={this.onClickDiv.bind(this)} />;

}

}

// tốt

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.onClickDiv = this.onClickDiv.bind(this);

}

onClickDiv() {

// Làm gì đó vui vẻ

}

render() {

return <div onClick={this.onClickDiv} />;

}

}

* Không nên dùng dấu "\_" đặt trước tên các hàm của Component

Lí do? Vì dấu gạch dước thi thoảng được dùng trong một số ngôn ngữ để biểu thị tính "private". Tuy nhiên, không giống các ngôn ngữ khác, trong JavaScript, mọi thứ đều là “public”. Cho dù bạn có cho dấu gạch dưới vào hay không nó vẫn là public, bất kể ý định của bạn.

// tệ

React.createClass({

\_onClickSubmit() {

// làm việc gì đó

},

// làm việc gì đó

});

// tốt

class extends React.Component {

onClickSubmit() {

// làm việc gì đó

}

// làm việc gì đó

}

* Phải trả về một giá trị trong hàm render. eslint: [react/require-render-return](https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/require-render-return.md)
* // tệ
* render() {
* (<div />);
* }
* // tốt
* render() {
* return (<div />);

}

## Cách sắp xếp hàm

* Các hàm trong class extends React.Component nên được viết theo thứ tự sau:

1. Các phương thức tĩnh static (không bắt buộc)
2. constructor
3. getChildContext
4. componentWillMount
5. componentDidMount
6. shouldComponentUpdate
7. componentWillUpdate
8. componentDidUpdate
9. componentWillUnmount
10. Hàm xử lí sự kiện như click hoặc submit onClickSubmit() & onChangeDescription()
11. Các hàm lấy dữ liệu cho hàm *render* chẳng hạn như getSelectReason() hay getFooterContent()
12. Các hàm render khác như renderNavigation() hay renderProfilePicture()
13. render

* Cách định nghĩa propTypes, defaultProps, contextTypes, ...
* import React from 'react';
* import PropTypes from 'prop-types';
* const propTypes = {
* id: PropTypes.number.isRequired, // Nếu id không đúng kiểu number, màn hình console sẽ hiện ra cảnh báo
* url: PropTypes.string.isRequired, // Nếu url không đúng kiểu string, màn hình console sẽ hiện ra cảnh báo
* text: PropTypes.string, // Nếu text không đúng kiểu string, màn hình console sẽ hiện ra cảnh báo
* };
* const defaultProps = {
* text: 'Hello World', // Gán giá trị mặc định cho text
* };
* class Link extends React.Component {
* static methodsAreOk() {
* return true;
* }
* render() {
* return <a href={this.props.url} data-id={this.props.id}>{this.props.text}</a>;
* }
* }
* Link.propTypes = propTypes;
* Link.defaultProps = defaultProps;

export default Link;

* Các hàm trong React.createClass nên được viết theo thứ tự sau: eslint: [react/sort-comp`](<https://github.com/yannickcr/eslint-plugin-react/blob/master/docs/rules/sort-comp.md>)

1. displayName
2. propTypes
3. contextTypes
4. childContextTypes
5. mixins
6. statics
7. defaultProps
8. getDefaultProps
9. getInitialState
10. getChildContext
11. componentWillMount
12. componentDidMount
13. componentWillReceiveProps
14. shouldComponentUpdate
15. componentWillUpdate
16. componentDidUpdate
17. componentWillUnmount
18. Hàm xử lí sự kiện như onClickSubmit() hay onChangeDescription()
19. Các hàm lấy dữ liệu cho phương thức *render* nhưgetSelectReason() hay getFooterContent()
20. Các hàm render khác như renderNavigation() hay renderProfilePicture()
21. render
22. Ví dụ

* Các bước để tạo và lấy dữ liệu cho 1 site:

1. Tạo router:

* Truy cập *config -> router.config.js*
* Tạo 1 đối tượng trong mảng router web, bao gồm các trường :path, name, component.
* Path: đường dẫn url của trình duyệt.
* Name: tên của router
* Component: đường dẫn vật lý của trang (trỏ ở thư mục pages)

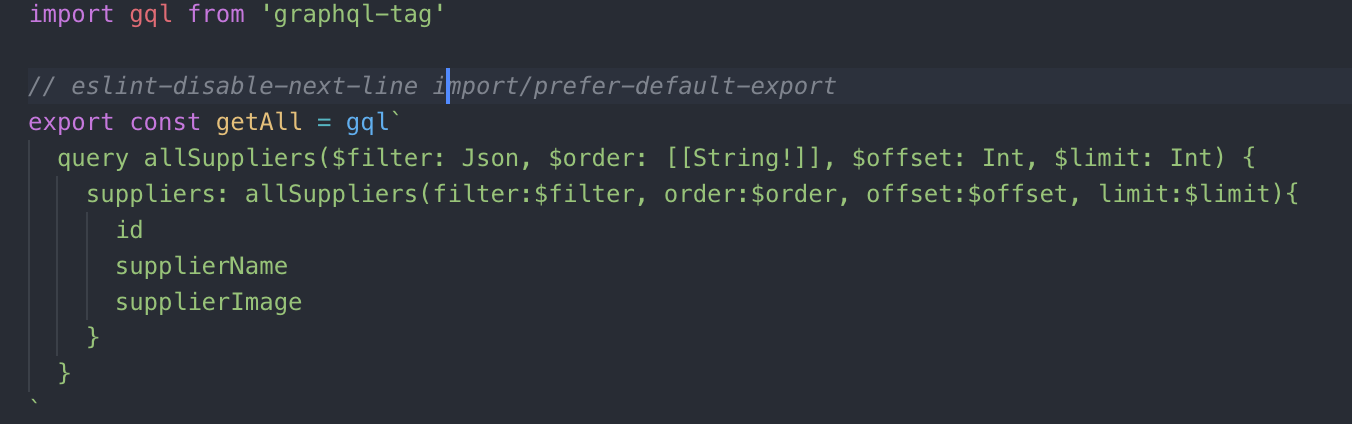


1. Tạo view cho site:

* Tại thư mục pages tạo 1 folder hoặc file phù hợp với đường dẫn vật lý ở *router.config.js.*Như ví dụ ở trên là tạo file *pages/index.js*
* File index.js sẽ là trang view chính của path: home. Những gì index render ra sẽ được view ở home.

1. Lấy dữ liệu cho trang.

* Có 2 kiểu lấy dữ liệu và 1 kiểu thay đổi dữ liệu.
  1. Lấy dữ liệu ở server:
  + Chỉ dùng kiểu lấy dữ liệu này ở trang chính (pages).
  + Định nghĩa gọi API ở folder *services:* Tại đây các định nghĩa được chia theo danh mục (product, category, …). Tạo nếu chưa có định nghĩa định dùng.



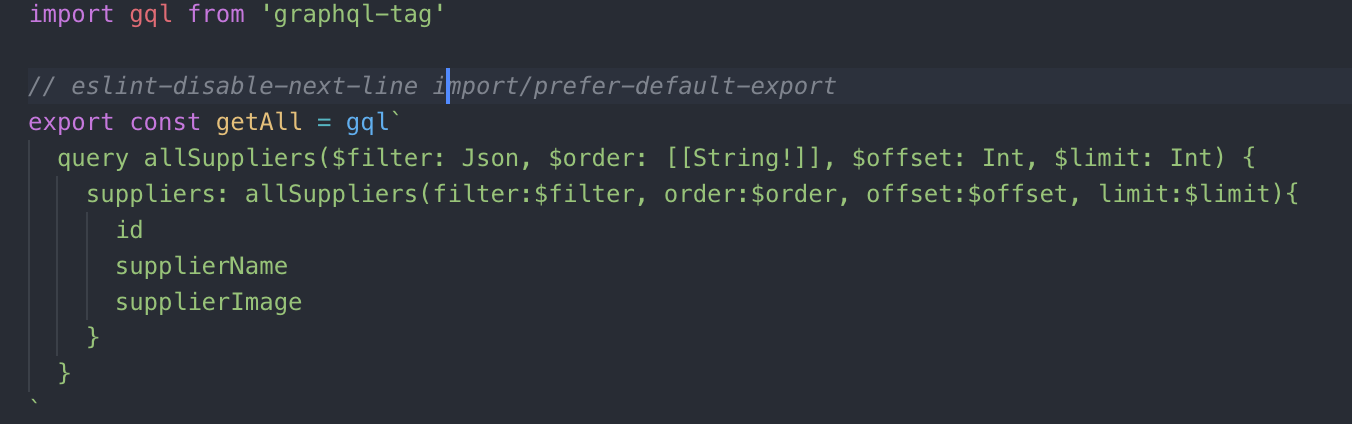
* + Định nghĩa phương thức gọi ở folder *controllers:* Các phương thức được chia theo địa chỉ sử dụng nó.(Dùng ở đâu thì phải tạo đúng theo đường dẫn định nghĩa, có thể tạo lặp hàm ở các file khác nhau)



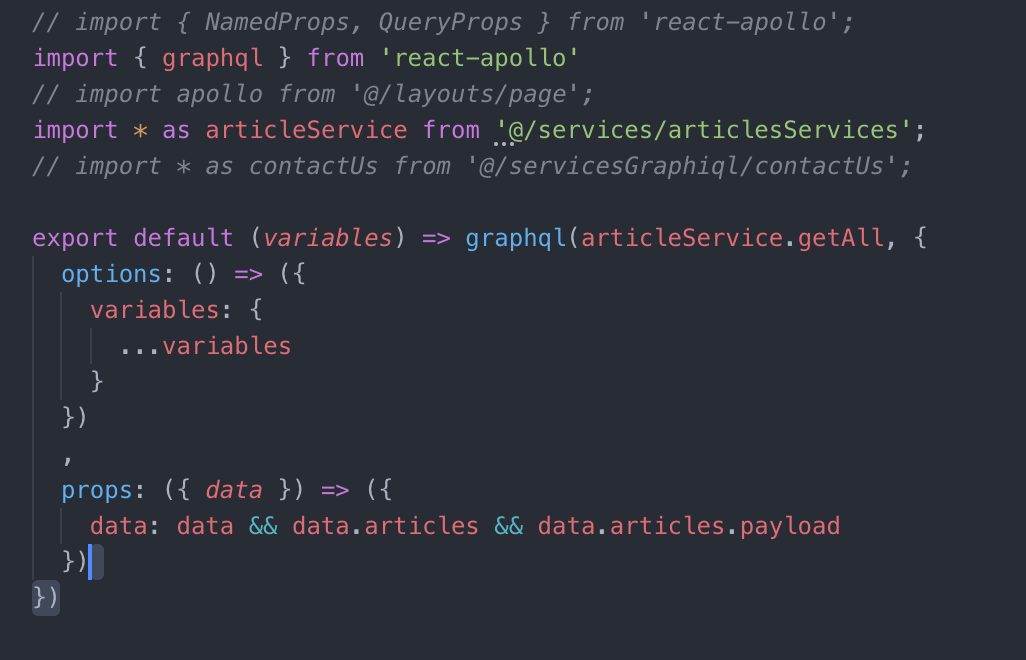
* + 2 đối tượng được truyền vào :apolloClient, variables. Và gọi đến định nghĩa API tại suplierService.getAll .
  + Sử dụng controllers tại *getIntialProps:* (pages/index.js)



* Import suppliers tại controllers
  1. Lấy dữ liệu ở client:
  + Dùng kiểu lấy dữ liệu này ở component phụ (componentWeb).
  + Định nghĩa gọi API ở folder *services:* Tại đây các định nghĩa được chia theo danh mục (product, category, …). Tạo nếu chưa có định nghĩa định dùng.



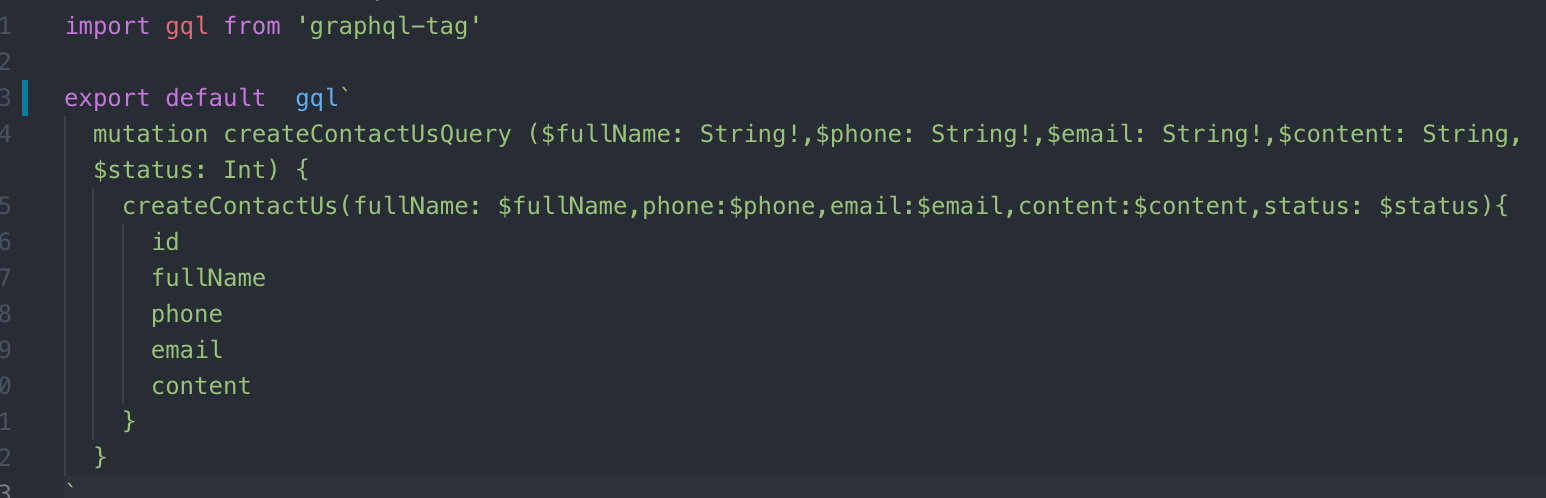
* + Định nghĩa phương thức gọi ở folder *controllers:* Các phương thức được chia theo địa chỉ sử dụng nó.(Dùng ở đâu thì phải tạo đúng theo đường dẫn định nghĩa, có thể tạo lặp hàm ở các file khác nhau)



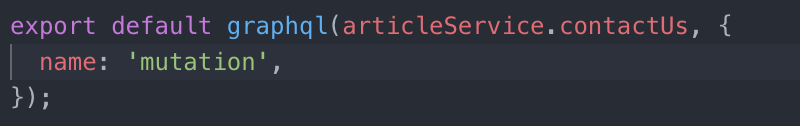
* + Đối tượng được truyền vào là: *variables*. Query đến *articleService.getAll*
  + Sử dụng phương thức gọi ở các componets nhúng tại folder componentWeb.



* Import articles tại controllers/components
  1. Thay đổi dữ liệu:
* Định nghĩa như việc lấy dữ liệu.



* Phương thức gọi.



* Thêm trường định danh nam để khi cần gọi đến chứ không auto get như lấy dữ liệu.
* Sử dụng chủ yếu ở component phụ.



* Import mutation controllers và tạo ở compose
* Sử dụng và truyền đi variables bằng trường name đã định nghĩa ở controllers.