Mục lục

[CHƯƠNG 1. Giới thiệu 2](#_Toc32671769)

[1.1. Công nghệ sử dụng trong dự án quản lý công việc 2](#_Toc32671770)

[1.2. Một số thuật ngữ cần chú ý 2](#_Toc32671771)

[CHƯƠNG 2. Mô hình phân quyền RBAC và cấu trúc cơ sở dữ liệu trong dự án quản lý công việc. 4](#_Toc32671772)

[2.1. Mô hình phân quyền RBAC 4](#_Toc32671773)

[2.2. Cơ sở dữ liệu của dự án Quản Lý Công Việc 4](#_Toc32671774)

[2.2.1. Cấu trúc dữ liệu 4](#_Toc32671775)

[2.2.2. Mô tả cho các Collection: 5](#_Toc32671776)

[2.3. Mô hình 12](#_Toc32671777)

[2.3.1. Mô tả: 12](#_Toc32671778)

[CHƯƠNG 3. Project Quản Lý Công Việc 14](#_Toc32671779)

[3.1. Coding convention 14](#_Toc32671780)

[3.1.1. MongoDB 14](#_Toc32671781)

[3.1.2. Functions 16](#_Toc32671782)

[3.1.3. NodeJS 16](#_Toc32671783)

[3.1.4. ReactJS 17](#_Toc32671784)

[3.2. Cấu trúc thư mục 18](#_Toc32671785)

[3.2.1. Cấu trúc của project 18](#_Toc32671786)

[3.2.2. Cấu trúc thư mục phần Server 18](#_Toc32671787)

[3.2.3. Cấu trúc thư mục phần Client 23](#_Toc32671788)

[CHƯƠNG 4. Cài đặt và cấu hình project 30](#_Toc32671789)

[CHƯƠNG 5. Tài liệu tham khảo 31](#_Toc32671790)

# Giới thiệu

## Công nghệ sử dụng trong dự án quản lý công việc

1. NodeJS, MongoDB – lập trình backend cho bên server
2. ReactJS, Redux – lập trình frontend cho bên client.

## Một số thuật ngữ cần chú ý

1. Database (DB): Nơi chứa các Collection – giống với cơ sở dữ liệu quan hệ thì chứa các bảng.
2. Collection: Là nhóm của nhiều Document trong MongoDB. Collection có thể được hiểu tương ứng như là 1 bảng trong cơ sở dữ liệu quan hệ (SQL). Các collection không nhất thiết phải định nghĩa các cột, các hàng hay kiểu dữ liệu trước như trong Cơ sở dữ liệu quan hệ vì tất cả thông tin được lưu dưới dạng document.
3. Document: Là 1 bản ghi thuộc 1 Collection. Các Document lần lượt bao gồm các trường tên và giá trị tương ứng.
4. JSON: Viết tắt của JavaScript Object Notation. Con người có thể đọc được ở định dạng văn bản đơn giản thể hiện cho các dữ liệu có cấu trúc.

A screenshot of a cell phone

Description automatically generated

Hình 1.1: Ví dụ về JSON

1. User: người dùng
2. Role: vai trò, chức danh, … của người dùng.
3. Resource: tài nguyên của hệ thống, có thể là các page của website, các button trên giao diện, các tài liệu biểu mẫu, …
4. Privilege: đặc quyền của từng Role ứng với từng loại tài nguyên.
5. Action: các loại hành động tương ứng với tài nguyên.

# Mô hình phân quyền RBAC và cấu trúc cơ sở dữ liệu trong dự án quản lý công việc.

## Mô hình phân quyền RBAC

RBAC (Role-Based Access Control): là một mô hình phân quyền dựa theo vai trò.

Trong đó:

Mỗi User sẽ có 1 hoặc nhiều Role.

Mỗi Role sẽ có 1 hoặc nhiều Privilege thực thi các Action tương ứng với một tài nguyên (Resource) nào đó trong hệ thống.

Tài nguyên có thể là một trang web nào đó, một thành phần giao diện, hoặc một tài nguyên do người dùng tạo ra khi hệ thống hoạt động.

Ví dụ: Ông Nguyễn Văn A có vai trò trong 1 công ty là Trưởng Phòng Kế Hoạch. Vai trò này cho phép ông có quyền truy cập vào **trang quản lý mẫu công việc**. Trong khi đó, chị Phạm Thị C, với vai trò là Nhân viên phòng kế hoạch, sẽ không vào được trang quản lý mẫu công việc đó. Ở đây, tài nguyên được xét đến là trang quản lý mẫu công việc.

Tương tự, dù cả hai cùng được vào một trang web nào đó (VD trang xem mẫu công việc), ông A sẽ thấy nút Edit cho từng mẫu công việc, nhưng chị C không thấy nút này. Ở đây, tài nguyên được xét đến là nút Edit.

Một ví dụ khác, với một tài liệu X nào đó, ông A có thể nhìn thấy, trong khi chị C không thể. Trong cùng trang xem danh sách tài liệu, tài liệu X hiển thị cho ông A thấy, và không xuất hiện với chị C. Sau này, khi ông A thôi chức trưởng phòng kế hoạch và do vậy không còn vai trò này nữa, tài liệu X sẽ không hiển thị cho ông A thấy nữa.

## Cơ sở dữ liệu của dự án Quản Lý Công Việc

### Cấu trúc dữ liệu

Database của hệ thống sẽ bao gồm các Collection chính sau đây:

* Company
* User
* Role
* UserRole
* RoleType
* Department
* Privilege
* Link
* Component
* Action

### Mô tả cho các Collection:

#### Company:

lưu trữ thông tin về các công ty sử dụng dịch vụ quản lý công việc của hệ thống.

|  |  |  |
| --- | --- | --- |
| **Thuộc tính** | **Kiểu giá trị** | **Mô tả** |
| **\_id** | *ObjectId* | ID của công ty |
| **name** | *String* | Tên đầy đủ của công ty |
| **short\_name** | *String* | Tên viết tắt của công ty |
| **description** | *String* | Mô tả về công ty |

#### User :

thông tin về tài khoản của người dùng.

|  |  |  |
| --- | --- | --- |
| **Thuộc tính** | **Kiểu giá trị** | **Mô tả** |
| **\_id** | *ObjectId* | ID của người dùng |
| **name** | *String* | tên của người dùng |
| **email** | *String* | địa chỉ email |
| **password** | *String* | mật khẩu |
| **company** | *ObjectId* | công ty mà người dùng làm việc |
| **active** | *Boolean* | trạng thái hoạt động của tài khoản người dùng |
| **status** | *Number* | đếm số lần tối đa nhập sai thông tin tài khoản khi đăng nhập |
| **reset**\_**password**\_**token** | *String* | token xác thực đổi mật khẩu khi quên |
| **token** | *Array String* | Mảng các token lưu trữ phiên đăng nhập của người dùng |

#### Role :

các role có trong hệ thống.

|  |  |  |
| --- | --- | --- |
| **Thuộc tính** | **Kiểu giá trị** | **Mô tả** |
| **\_id** | *ObjectId* | ID của role |
| **name** | *String* | tên của role |
| **company** | *ObjectId* | Công ty có role này |
| **type** | *ObjectId* | Loại role (lưu trong collection RoleType) |
| **parents** | *Array ObjectId* | Là mảng các role là cha của role hiện tại  Role hiện tại, ngoài các quyền của chính nó với một loại tài nguyên nào đó, sẽ kế thừa thêm quyền của những role cha với các tài nguyên khác.  Chỉ cho phép kế thừa trực tiếp, không cho phép kế thừa gián tiếp.  VD: Có 3 Role, Role 2 kế thừa Role 1, và Role 3 kế thừa Role 2 như sau:  Role 1🡨 Role 2🡨 Role 3.  Nếu Role 1 có quyền xem tài liệu X. Role 2 dù không được liên kết với tài liệu X (không có quyền gì), nhưng vì kế thừa trực tiếp Role 1, Role 2 vẫn có quyền xem tài liệu X. Giả sử Role 3 không có liên kết gì tới tài liệu X, do Role 3 không kế thừa trực tiếp Role 1, nên nó sẽ không có quyền xem tài liệu X |

#### UserRole :

chứa thông tin về mối liên hệ *Many to Many( nhiều-nhiều)* giữa **User** và **Role**. Một mối liên hệ nhiều – nhiều giữa nhiều User với nhiều Role sẽ bao gồm nhiều cặp thông tin bao gồm Id của user sẽ tương ứng với Id của role nào.

|  |  |  |
| --- | --- | --- |
| **Thuộc tính** | **Kiểu giá trị** | **Mô tả** |
| **userId** | *ObjectId* | ID của user |
| **roleId** | *ObjectId* | ID của role |

#### RoleType :

các loại role có trong hệ thống.

|  |  |  |
| --- | --- | --- |
| **Thuộc tính** | **Kiểu giá trị** | **Mô tả** |
| **\_id** | *ObjectId* | ID loại role |
| **name** | *String* | Tên loại ( kiểu ) role |

Trong hệ thống được định nghĩa sẵn 3 loại role đó là: RoleAbstract, RoleChucDanh và RoleTuTao.

**1-RoleAbstract**: Là loại role được thiết kế riêng để phân quyền cho giao diện frontend của hệ thống. *Tất cả các role thuộc loại RoleAbstract là không thể xóa*. Mặc định sẽ có 5 role thuộc loại RoleAbstract đó là:

* **System Admin** – là role quản trị viên của toàn bộ hệ thống. Hệ thống có thể phục vụ đồng thời cho nhiều công ty khác nhau. **System Admin** không không tham gia vào hoạt động nghiệp vụ bất kì công ty nào. **System Admin** chỉ có nhiệm vụ khởi tạo các dữ liệu ban đầu cho một công ty mới (khởi tạo dữ liệu về công ty, và tạo tài khoản SuperAdmin cho công ty đó, …)
* **Super Admin**: là role quản trị cấp cao nhất của 1 công ty. Trong công ty đó, chỉ có duy nhất 1 tài khoản user X được gắn role là **super admin**. Tài khoản X này được tạo khi **System Admin** tạo mới 1 công ty. Không thể gán role **Super Admin** cho bất kỳ tài khoản nào khác ngoài tài khoản X này, cũng không thể bỏ role này khỏi tài khoản X. Tài khoản X này cũng không thể bị xóa đi. Role **Super Admin** kế thừa **Admin**, nên sẽ có mọi quyền của role **Admin**.
* **Admin**: role quản trị của công ty. **Super Admin** hoặc một **Admin** bất kỳ có thể gán role này cho tài khoản Y bất kỳ trong công ty đó. Không thể xóa role **Admin**, nhưng có thể xóa (deactivate) tài khoản bất kỳ có role này, cũng như có thể thu hồi role **Admin** cho một tài khoản bất kỳ.
* **Dean**: Trưởng một đơn vị bất kỳ trong công ty, sẽ vào được nhiều trang thông tin hơn, cũng có nhiều menu chức năng hiển thị hơn so với **Vice Dean**, **Employee**. Việc phân quyền này có thể linh động cho từng công ty, được quản lý bởi **Admin**.
* **Vice** **Dean**: phó trưởng một đơn vị trong công ty
* **Employee**: nhân viên của đơn vị bất kỳ trong công ty

**2-RoleChucDanh**: là loại role đại diện cho một chức vụ, chức danh của người dùng trong các phòng, ban, bộ phận, khối (được gọi tổng quát là một **đơn vị**) trong một công ty. Mỗi một **RoleChucDanh** phải luôn đi kèm với một **Department** (*đơn vị, phòng ban, …*) nào đó. Với một **RoleChucDanh,** người dùng sẽ có quyền tương ứng với một tài nguyên mà trong phạm vi **Department** đó có thể thực hiện*.* Ví dụ, một tài liệu nào đó sẽ chỉ được xem bởi nhân viên phòng Hành chính và chỉ được edit bởi trưởng phòng phòng Hành chính. Nhân viên phòng khác hoàn toàn không biết đến sự tồn tại của tài liệu này.

Khi tạo một đơn vị mới, ví dụ phòng kinh doanh, hệ thống tự động tạo 3 role chức danh với quan hệ kế thừa như sau: *Chức danh trưởng phòng kinh doanh* 🡨 *Chức danh phó phòng kinh doanh* 🡨 *Chức danh nhân viên phòng kinh doanh*. Người dùng được nhập vào tên cụ thể của 3 loại chức danh này. Trong đó, lưu ý: trưởng phòng kinh doanh sẽ kế thừa 5 role khác (3 role abstract, 2 role chức danh), đó là: **Dean**, **Vice Dean**, **Employee**, phó phòng kinh doanh, nhân viên phòng kinh doanh. Tương tự, phó phòng kinh doanh kế thừa **Vice Dean**, **Employee**, nhân viên phòng kinh doanh. Cuối cùng, nhân viên phòng kinh doanh chỉ kế thừa role **Employee**. Điều này cho phép Trưởng phòng kinh doanh xem được mọi tài nguyên mà phó phòng hoặc nhân viên phòng kinh doanh được xem.

Ngoài ra, cần lưu ý trường hợp sau. Giả sử trong cơ cấu tổ chức của 1 công ty, Phòng kinh doanh trực thuộc trong Khối Văn phòng thì người đứng đầu phòng kinh doanh sẽ có 2 role, một role là **Trưởng phòng kinh doanh,** và role kia là **nhân viên Khối Văn phòng**.

**3-RoleTuTao**: là role được tạo thêm nhằm mục đích phân quyền một cách cụ thể với một tài nguyên nào đó hoặc áp dụng cho một nhóm người cụ thể nào đó. Những Role thuộc vào nhóm **RoleTuTao** không nhiết thiết phải đi kèm hay có ràng buộc với một đơn vị, phòng ban nào mà có thể hoàn toàn độc lập trong hệ thống.

#### Department :

lưu thông tin về các đơn vị/phòng ban của 1 công ty.

|  |  |  |
| --- | --- | --- |
| **Thuộc tính** | **Kiểu giá trị** | **Mô tả** |
| **\_id** | *ObjectId* | ID của đơn vị |
| **name** | *String* | Tên đơn vị |
| **company** | *String* | Công ty chứa đơn vị |
| **description** | *String* | mô tả về đơn vị |
| **dean** | *ObjectId* | Trưởng đơn vị |
| **vice\_dean** | *ObjectId* | Phó đơn vị |
| **employee** | *ObjectId* | Nhân viên trong đơn vị |
| **parent** | *ObjectId* | Đơn vị cha của đơn vị hiện tại |

Mỗi một đơn vị sẽ có 3 RoleChucDanh được tạo ra, như mô tả ở phần các loại Role*.*

#### Privilege :

chứa phân quyền của từng role ứng với một tài nguyên cụ thể.

|  |  |  |
| --- | --- | --- |
| **Thuộc tính** | **Kiểu giá trị** | **Mô tả** |
| **resourceId** | *ObjectId* | ID của tài nguyên |
| **resourceType** | *String* | Loại tài nguyên. Hiện tại có nhưng loại tài nguyên là Link, Component. |
| **roleId** | *ObjectId* | Role tương ứng có quyền với tài nguyên |
| **action** | *ObjectId* | Đây là thuộc tính được thêm vào để xác định rõ hơn mối quan hệ giữa role (roleChucDanh, roleTuTao) với cái tài nguyên không phải giao diện (link, component) như tài liệu, biểu mẫu, công văn, ...  Khác với tài nguyên link, component, các loại tài nguyên khác (tài liệu, biểu mẫu, công việc, ...) cần rõ ràng hơn về các hành động role được phép thực hiện như là: xem, thêm, sửa, xóa.  Đối với RoleAbstract, khi liên kết với tài nguyên giao diện (link, component) giá trị mặc định cho trường này sẽ là NULL. Điều này không phải có nghĩa là RoleAbstract không có quyền với tài nguyên giao diện mà là không cần chia rõ ràng xem, thêm, sửa, xóa với tài nguyên giao diện. Giao diện là do lập trình viên lập trình, người dùng cuối không thể xóa/sửa/thêm được. VD với một link, Admin sẽ được truy cập để mở ra trang tương ứng. Như vậy là đủ (gần như chỉ cần quyền xem) |

Bản chất giữa Role và các loại tài nguyên (Resource) có mối quan hệ nhiều – nhiều. Collection Privilege này có tính chất như một bảng trung gian có chức năng giống với collection UserRole ở trên. Privilege sẽ đảm nhận nhiệm vụ lưu các cặp dữ liệu Role – Resource (tài nguyên ở đây có thể là các trang - Link, các button trên giao diện của người dùng - Component, các biểu mẫu công việc – TaskTemplate, …)

#### Link :

lưu trữ thông tin của các page của website hệ thống quản lý công việc.

|  |  |  |
| --- | --- | --- |
| **Thuộc tính** | **Kiểu giá trị** | **Mô tả** |
| **\_id** | *ObjectId* | ID trang page |
| **url** | *String* | URL của page |
| **company** | *ObjectId* | Công ty chứa page này |
| **description** | *String* | Mô tả |
| **components** | *Array ObjectId* | Các thành phần UI, UX được phân quyền theo từng role trên page này. |

#### Component :

những thành phần UI, UX trên từng page.

|  |  |  |
| --- | --- | --- |
| **Thuộc tính** | **Kiểu giá trị** | **Mô tả** |
| **\_id** | *ObjectId* | ID trang page |
| **name** | *String* | Mô tả |
| **company** | *ObjectId* | Công ty chứa page này |
| **description** | *String* | Mô tả |

#### Action :

các hành động tương ứng của từng role với mỗi loại tài nguyên khác nhau. Ví dụ Trưởng Phòng Kế Hoạch sẽ có thể action (*xem, sửa, xóa, kích hoạt*,…) 1 loại tài liệu biểu mẫu công việc nào đó.

|  |  |  |
| --- | --- | --- |
| **Thuộc tính** | **Kiểu giá trị** | **Mô tả** |
| **\_id** | *ObjectId* | ID trang page |
| **name** | *String* | Tên của hành động |
| **see** | *Boolean* | Cho phép nhìn thấy tài nguyên |
| **open** | *Boolean* | Cho phép mở tài nguyên |
| **edit** | *Boolean* | Cho phép chỉnh sửa tài nguyên |
| **delete** | *Boolean* | Cho phép xóa tài nguyên |
| **enable** | *Boolean* | Cho phép kích hoạt hoạt động cho tài nguyên |
| **disable** | *Boolean* | Cho phép khóa tài nguyên |

## Mô hình

A close up of a map

Description automatically generated

Tham khảo cách mô hình hóa csdl trong MongoDB

<https://docs.mongodb.com/manual/core/data-model-design/>

### Mô tả:

Hệ thống quản lý công việc sẽ quản lý cho nhiều công ty cùng một lúc. Cần phải biết được các đối tượng thuộc về công ty nào. Vì vậy, mỗi một user, role, department, resource (link, component, …) sẽ có một thuộc tính là company – để xác định user, role, department, resource đó thuộc về công ty nào.

Giữa Link (1 page) và Component (Thành phần của giao diện) có mối liên hệ One to Many – 1 trang sẽ bao gồm nhiều thành phần (button, form, …) thì mối liên hệ này sẽ được biểu diễn : trong collection Link sẽ có 1 trường là components – định danh là một mảng các Id của component tương ứng với page này.

Giữa User và Role có mối liên hệ là một user có thể có nhiều role và một role có thể là role của 1 hoặc nhiều user. Điều này khá giống với quan hệ n-n (trong cơ sở dữ liệu quan hệ). Tuy nhiên trong MongoDB hiện tại chưa có khái niệm (Many to Many) nên để thể hiện được quan hệ Many to Many như của cơ sở dữ liệu quan hệ thì chúng ta sẽ xây dựng collection trung gian UserRole để lưu giữ mối liên hệ này. Cụ thể, mỗi một bản ghi trong UserRole sẽ là một cặp giá trị (user, role). Trong model User (code phần server) có một trường ảo là roles – sẽ truy xuất tất cả những role mà user này có. Ngược lại với model Role sẽ có một trường ảo là users – sẽ truy xuất tất cả những user có role này.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Tương tự với mối liên hệ giữa role và các loại tài nguyên cũng là Many to Many (collection trung gian là Privilege) thì sẽ có các trường ảo cho thuộc tính tương ứng để truy xuất dữ liệu giữa hai bên. Ví dụ như 1 role có quyền với nhiều trang (có trường ảo là links), có quyền với nhiều component (có trường ảo là components) như hình mô tả cấu trúc của Role như ở trên. Tương ứng với mỗi một loài nguyên sẽ đi kèm với một action cấp phép cho role có quyền với tài nguyên được phép làm những gì với tài nguyên đó (xem, sửa, xóa, …).

# Project Quản Lý Công Việc

## Coding convention

### MongoDB

#### Quy tắc chung

Sử dụng 4 space để lùi đầu dòng, điều này áp dụng cho tất cả những mã code của MongoDB và cho tất cả các đối tượng sử dụng mongoDB (Queries, documents)

Luôn có một khoảng trắng sau dấu hai chấm

Dấu phẩy ở cuối

Nếu chia thành phần của mỗi đối tượng hoặc mảng thành nhiều dòng khác nhau thì hãy chia mỗi thành phần thành một dòng, dấu ngoặc nhọn đóng phải đi theo sau thành phần cuối cùng (ngoại trừ tập hợp)

Tất cả các tên đều không được chứa khoảng trắng

Không có dấu gạch dưới ở giữa tên (database, collection, fields).

Tên của các collection, variables, properties và function sử dụng quy tắc lowerCamelCase. Các biến kí tự đơn hoặc viết tắt không phổ biến thì không được sử dụng.

A screen shot of a monitor

Description automatically generated

Hình 3.1 Ví dụ về coding convention trong MongDB

#### Đặt tên Database

Tên cơ sở dữ liệu đặt theo tên của dự án

Sử dụng UpperCamelCase

Tên cơ sở dữ liệu không được dài quá (ít hơn 64 bytes)

Tên cơ sở dữ liệu phải luôn phân biệt chữ hoa chữ thường

Tên cơ sở dữ liệu không được chứa các ký tự đặc biệt “/, \, ., “, \*, <, >, :, |, ?, $”. Đồng thời cũng không được chứa các khoảng trắng hoặc chuỗi rỗng

#### Đặt tên Collection

Sử dụng UpperCamelCase

Tên collection không được chứa khoảng trắng hoặc là chuỗi rỗng

#### Đặt tên Field

Sử dụng LowerCase

Không sử dụng dấu gạch dưới trong tên trường ngoại trừ id

Không được sử dụng dấu chấm hoặc chuỗi rỗng và không bắt đầu bằng $

### Functions

Nếu chia các thành phần thành nhiều dòng thì mỗi thành phần sẽ để một dòng

Sử dụng dấu tab để lùi đầu dòng với mỗi thành phần

### NodeJS

#### Quy tắc chung

Sử dụng 4 space để lùi đầu dòng và không trộn lẫn tab với khoảng trống khi căn lề

Không để khoảng trắng cuối mỗi tệp .js

Tối đa 80 kí tự trên một dòng

Sử dụng dấu ngoặc đơn, ngoại trừ khi viết JSON

Ví dụ đúng: var foo = ‘bar’;

Dấu ngoặc mở trên cùng một dòng với statement

Kết thúc câu lệnh khai báo với dấu chấm phẩy trên cùng một dòng.

#### Đặt tên

Tên biến và hàm bao gồm nhiều từ thì kí tự đầu mỗi từ viết hoa trừ từ đầu tiên. ( VD: departmentName )

Tên lớp viết hoa chữ cái đầu tiên của các từ. Các chữ còn lại viết thường. ( VD: ViceDean,…)

Tên của hằng thì tất cả chữ cái viết hoa (VD: LOGGED, AUTHENTICATION,… )

### ReactJS

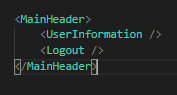
#### Quy tắc chung

Chia Component nhỏ nhất có thể

Mỗi Component lưu riêng vào 1 file (ưu tiên dạng *.jsx* với các file giao diện).

Sử dụng Composition để mở rộng chức năng (thông qua props thì một component cha có thể render ra một hoặc nhiều componet con)

JSX đa dòng: Mỗi một component nên được để riêng trên một dòng. Trước khi viết thụt lùi một dấu Tab.



Hình 3.2 JSX đa dòng

#### Đặt tên

**Tên tệp**: Sử dụng UpperCamelCase cho tên tệp

**Tên tham chiếu**: Sử dụng UpperCamelCase cho các thành phần React và lowerCamelCase cho các thể hiện của chúng

Ví dụ:

// sai

import reservationCard from './ReservationCard';

// đúng

import ReservationCard from './ReservationCard';

// sai

const ReservationItem = <ReservationCard />;

// đúng

const reservationItem = <ReservationCard />;

**Tên component**:

Tên component phải duy nhất và thể hiện được vai trò của nó

Ví dụ: SideBar, ChatConversationName, …

Tên của component được đặt tên theo đường dẫn

Ví dụ: component/User/List -> UserList

**Tên props**: Tránh dùng tên prop components Dom cho các mục đích khác

Ví dụ:

Sai: <MyComponent style=”fancy”/>

Đúng: <MyComponent variant=”fancy”/>

## Cấu trúc thư mục

### Cấu trúc của project

Cấu trúc của project Quản lý công việc bao gồm hai phần chính:

Server: backend

Client: frontend

### Cấu trúc thư mục phần Server

Trong thư mục server có chứa mã nguồn cho phần backend của hệ thống, các thành phần trong thư mục server bao gồm:

logs: chứa file ghi lại lịch sử tương tác giữa client và server.

middleware: xác thực người dùng từ request và kiểm tra, nếu thông tin là chính xác thì sẽ chuyển tiếp cho server thực hiện yêu cầu mà client gửi đến. Ngược lại nếu xác thực người dùng không hợp lệ thông báo lỗi yêu cầu của client không được server xử lý và trả về thông báo lỗi cho client.

models: định nghĩa các collection cho cấu trúc dữ liệu của hệ thống.

seed: chứa các file tạo dữ liệu gốc khi build hệ thống, hoặc lập trình viên có thể khởi tạo các dữ liệu test tại đây.

modules: chứa các file module – tương ứng với từng chức năng, dịch vụ của hệ thống. Mỗi một module khi được xây dựng sẽ nằm trong thư mục này. Trong thư mục modules có chứa sẵn 1 thư mục con là “ *\_sample-module “* – đây là một thư mục module mẫu. Trong “\_sample-module “ bao gồm 4 file chính đó là :

README.txt: file mô tả về chức năng của module, cấu trúc của module và sự liên quan của module với các module khác.

route.js: định nghĩa các đường định tuyến, các API cho từng dịch vụ của hệ thống mà người dùng (client sẽ gọi để yêu cầu sử dụng dịch vụ. Có thể lồng thêm middleware nhằm thực hiện nhiệm vụ xác thực người dùng trước khi request được chuyển tiếp sang cho controller.

controller.js: nhận các request của người dùng và gọi đến các dịch vụ (service) để xử lý dữ liệu và trả về (response) cho người dùng (client).

service.js: chứa các service (dịch vụ) – nhận dữ liệu đầu vào, xử lý yêu cầu và đưa ra kết quả đầu ra. Mỗi một hàm trong trong service chỉ nên thực hiện một chức năng duy nhất.

Ví dụ: trong service của module user có hàm getById – lấy thông tin user theo Id : thì hàm này chỉ nên thực hiện một nhiệm vụ duy nhất là truy xuất thông tin về user theo Id.

index.js: file chạy của server.

package.json: chứa tên các thư viện cần được cài đặt trong project.

Ví dụ tạo một module mới cho phần server.

Bước 1: Tạo một thư mục cho module cần xây dựng trong thư mục server/modules. Cấu trúc của module sẽ giống với module mẫu \_sample-module. Ngoài ra trong quá trình lập trình có thể bổ sung thêm các file khác (ngoài 4 file mẫu trong thư mục \_sample-module) . Ví dụ: validation – nhằm mục đích kiểm tra dữ liệu đầu vào do client người đến trước khi cho xử lý. Lưu ý rằng với những module có khối lượng lớn thì ta sẽ chia nhỏ thành những module con và những module con này sẽ có cấu trúc giống với thư mục \_sample-module.

Bước 2: Viết mã nguồn cho file service.js để xử lý yêu cầu dịch vụ trong module. Định nghĩa rõ ràng các tham số đầu vào cho mỗi một chức năng trong file service.

Bước 3: Viết mã nguồn cho file controller.js . Tại đây controller nhận dữ liệu đầu vào qua request (req) được gửi đến từ bên client và truyền đến cho service để thực hiện xử lý dữ liệu. Controller có thể gọi đến các dịch vụ của chính module hiện tại hoặc có thể gọi thêm các service từ các module khác. Cuối cùng là trả về dữ liệu cho người dùng qua response (res).

Bước 4: Viết mã nguồn cho file route.js. Tạo ra các API tương ứng với từng chức năng của module.

Bước 5: Import route của module vừa xây dựng vào file chạy (index.js) của bên server.

Ví dụ: Xây dựng chức năng lấy tất cả thông tin về các user trong hệ thống trong module quản lý user .

Vì module quản lý user là một module con trong module cha là quản lý hệ thống của SuperAdmin của 1 công ty 🡪 tạo cấu trúc thư mục như hình:

A screenshot of a video game

Description automatically generated

Tiếp theo, trong file user.service.js tạo hàm get() – lấy thông tin của tất cả user trong công ty được chọn. Hàm get() nhận đầu vào là Id của công ty được chọn và search trong collection User tất cả những user có company bằng Id của công ty được truyền vào. Cụ thể: find() sẽ tìm kiếm tất cả nhưng document trong collection User với điều kiện là company là tham số nhận được, select() sẽ tùy chọn những thuộc tính của user sẽ được lấy ra trong quá trình truy vấn, populate – truy vấn tất cả các roles mà user có thông qua trường ảo roles được thiết lập mối quan hệ Many to Many giữa collection User và Role, cộng với đó là thông về thông tin mà user đó đang làm việc. Kết quả trả về là một mảng những user.

A screenshot of a computer screen

Description automatically generated

Tiếp theo là tạo controller trong file user.controller.js gọi đến service get – lấy thông tin tất cả user trong công ty. Trong khối try-catch thì try sẽ làm nhiệm vụ gọi service để xử lý dữ liệu và trả về cho client với mã là *200*. Nếu phát hiện có lỗi khối catch sẽ bắt lỗi đó và trả về cho client với mã lỗi là 400.

A screenshot of a computer screen

Description automatically generated

Tạo API cho chức năng lấy tất cả thông tin user trong file user.route.js. Bởi vì nếu như muốn gọi đến API truy vấn thông tin của tất cả user trong công ty thì bắt buộc phải có một middleware để check xem yêu cầu này có được gửi bởi một tài khoản đã được xác thực – thông qua middleware auth.

A screenshot of a computer screen

Description automatically generated

Cuối cùng là import route của module quản lý user vào file index.js

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer screen

Description automatically generated

Để test dữ liệu nhận được khi gọi API , ta sử dùng phần mềm POSTMAN để gọi đến API đó và kết quả nhận được sẽ là một mảng các users lưu theo cấu trúc JSON hoặc trả về thông báo lỗi nếu có lỗi xảy ra.

Như vậy là chúng ta đã viết xong 1 API – lấy thông tin tất cả user của một công ty.

### Cấu trúc thư mục phần Client

Thư mục client chứa mã nguồn cho phần frontent của hệ thống, cấu trúc như hình dưới đây:

A screenshot of a cell phone

Description automatically generated

Cấu trúc của project về cơ bản giống với frame của 1 project ReactJS . Tuy nhiên có sự khác biệt đó là ở trong thư mục src- cấu trúc mã nguồn phân chia theo từng module tương ứng với các module của bên server. Tổ chức và xử lý dữ liệu nhận về từ server được xử lý bằng Redux – trong thư mục src.Hai phần quan trọng nhất là :

public: chứa các file css,js, image,vv… chung cho cả project.

src: *chứa mã nguồn chính cho phần frontend*

common-components: chứa các component về giao diện được tái sử dụng – có thể được gọi đến từ các module (ví dụ: modal, form, thẻ input, …)

layouts: chứa các file giao diện của phần layout – bố cục của 1 trang trên website. Bao gồm Header, SideBar , Content và Footer.

modules: chứa các module tương ứng với các module của bên server. Mỗi một module của bên client sẽ bao gồm hai phần chính:

components – giao diện của module, chứa các file *.jsx*

redux – xử lý dữ liệu nhận về từ server ,bao gồm 4 file:

action.js – thực hiện lời gọi đến các hành động mà người dùng yêu cầu

constants.js – định nghĩa tên cho các hành động

reducer.js – xử lý dữ liệu sau khi xử lý và cập nhật vào trong store

service.js – xử lý dữ liệu tương ứng với hành động mà người dùng yêu cầu ( gọi API , gửi request, nhận dữ liệu trả về từ server và xử lý ).

react-routes: định nghĩa các route ứng với từng trang của website

redux: là nơi tập hợp state chung , kho store của redux.

translation.js : file đa ngôn ngữ - hiện tại đã có tiếng Việt và tiếng Anh

Ví dụ cách tạo một module mới cho phần client

Ở phần server chúng ta đã xây dựng một API cho chức năng liệt kê tất cả các user của một công ty. Vậy nên ở phần này chúng ta sẽ làm phần giao diện và gọi API cho chức năng đó ở bên client.

Mã nguồn cho chức năng này nằm trong module quản lý người dùng:

A screenshot of a computer screen

Description automatically generated

Đầu tiên,vào thư mục redux, trong file service.js ta sẽ viết chức năng lấy thông tin của tất cả người dùng trong một công ty:

A screenshot of a computer

Description automatically generated

Tương ứng với API ở bên server là */user* với ta truyền vào option có định dạng là :

{

**url** – đường dẫn tương ứng với API,

**method** – phương thức gọi ( cụ thể trường hợp ở đây là GET ),

headers – phần headers của http Request ta gửi thêm các dữ liệu như jwt token, browser-finger để có yêu cầu được middleware của server xác thực và cho tiến hành thực hiện gọi service bên server. Trong token – nhận được sau khi login và lưu trong localStorage có chứa các thông tin về user đã được mã hóa,bao gồm như tên, email, roles, ... và đi kèm là thông tin về công ty mà user đó đang làm việc – Id của công ty dùng để thực hiện lệnh truy vấn sẽ được lấy ra từ trong token này do client gửi lên.

**data** – với những method của yêu cầu thêm, sửa dữ liệu như POST, PATCH thì đây là phần chứa dữ liệu mà client muốn gửi đến server để chỉnh sửa data trong DB. Ở đây GET không yêu cầu chỉnh sửa mà chỉ trích xuất dữ liệu nên phần này là không cần thiết.

}

Có hai cách để client có thể thực hiện được lời gọi đến API của server là sử dụng axios hoặc fetch đều được. Ở đây chúng ta sử dụng axios.

Sau khi viết xong service ta chuyển đến file constants.js để định nghĩa hành động cho yêu cầu lấy tất cả user trong công ty.

A screenshot of a computer

Description automatically generated

Tiếp đến ta chuyển qua file action.js để viết lời gọi cho hành động này:

A screenshot of a computer

Description automatically generated

Tiếp theo,chúng ta chuyển qua file reducer.js và viết mã nguồn xử lý tương ứng với hành động được yêu cầu trong fle action.js

A screenshot of a computer

Description automatically generated

Sau khi viết xong reducer, ta vào thư mục src/redux để import state của user – lưu trữ data nhận được từ server vào trong store trong file combine-reducers.js

A screenshot of a computer screen

Description automatically generated

Cuối cùng là chúng ta sẽ đi vào code giao diện , lấy dữ liệu từ api và đổ ra màn hình. Hiển thị danh sách các user ta sẽ làm ở component <ManageUserTable />

Sau khi đã viết xong code cho phần gọi API, việc còn lại là ta gọi hàm chạy đó bên trong component này và hiển thị ra.

Đầu tiên ta khai báo UserActions để gọi action get – lấy tất cả user .

A screenshot of a computer screen

Description automatically generated

Tiếp theo đó ta sẽ kết nối đến **store** của redux, gọi đến hàm get của UserActions.

A screenshot of a computer

Description automatically generated

Tiếp theo ta sẽ gọi hàm get trong componentDidmout để load dữ liệu.

A screenshot of a computer screen

Description automatically generated

Dữ liệu trả về sẽ được lưu vào trong user.list – là 1 mảng danh sách các user trong công ty được trả về từ server. Ta sẽ lấy được dữ liệu này thông của qua props.

A screenshot of a computer screen

Description automatically generated

Cuối cùng ta chỉ việc liệt kê tất các các phần tử trong mảng list ra giao diện màn hình.

A screenshot of a computer screen

Description automatically generated

Ta có thể kiểm tra kết quả gọi API trên giao diện thông qua công cụ Redux Dev Tool. Khi check giá trị của user.list nếu như là 1 mảng dữ liệu ví dụ như trong hình thì có nghĩa API đã được gọi và client đã nhận được dữ liệu.

A screenshot of a computer

Description automatically generated

Kết thúc! 😊

# Cài đặt và cấu hình project

* + **Bước 1: Clone project QLCV**

*git clone https://github.com/VNISTResearch/qlcv.git*

* + **Bước 2: Cấu hình cho server**
    - Chạy lệnh ***npm install ,*** nếu xuất hiện cảnh báo về phiên bản cũ, chạy lệnh : ***npm audit fix*** để cập nhật.
    - Tạo file *.env* với cấu trúc giống như trong file *.env.example* bằng lệnh : **cp .env.example .env** , sau đó sửa nội dung của file .env :
      * DATABASE: cơ sở dữ liệu của bạn. Ví dụ: ***mongodb://localhost/qlcv***
      * TOKEN\_SECRET: chuỗi kí tự đặc biệt (đặt tùy ý – ví dụ: **congtyvnist** ) dùng để mã hóa token JWT gửi về cho client.
      * PORT: cổng dùng để chạy server ( ví dụ : **8000** ).
      * WEBSITE : link của website bên phía client ( ví dụ : **http://localhost:3000** )
    - Di chuyển đến thư mục **server/seed** và chạy lệnh

***node initDatabase.js*** để thiết lập các dữ liệu mặc định trong cơ sở dữ liệu

* + - Chạy server: lệnh chạy server có thể được cấu hình lại trong file package.json, nhưng mặc định sẽ có hai kiểu khởi chạy.
      * ***npm run dev*** : server sẽ khởi động lại mỗi khi mã nguồn có sự thay đổi mới. ( Điều kiện: máy đã cài đặt gói **nodemon**, nếu chưa cài đặt chạy lệnh ***npm install nodemon*** )
      * ***npm start*** : server sẽ không khởi lại khi mã nguồn có sự thay đổi.
  + **Bước 3:Cấu hình cho client**
    - Tương tự di chuyển vào thư mục client và chạy lệnh: ***npm install***
    - Cài đặt bower cho máy tính bằng lệnh : ***npm install bower -g***
    - Sau khi cài đặt xong bower, chạy lệnh ***bower install*** để download các file css, js và image cần thiết cho bên client. Sau khi tải xong trong thư mục public sẽ xuất hiện hai thư mục **lib** – chứa các file css,js… cho giao diện của client.
    - Tạo file cấu hình bên client , di chuyển vào thư mục **src,** chạy lệnh : **cp env.example.js env.js** để tạo ra file **env.js** và chỉnh sửa nội dung như yêu cầu trong file. Ví dụ:

A screen shot of a clock

Description automatically generated

* + - Chạy client : ***npm start***

\*Nếu có lỗi trong quá trình cài đặt, cấu hình hay chạy project, vui lòng điền thông tin về lỗi vào đây:

<https://docs.google.com/spreadsheets/d/1zSrSCBM7nXXBDrjveTh58IdIRaDlvXuPy_nhDoh-MoU/edit?usp=sharing>

# Tài liệu tham khảo

https://www.tutorialspoint.com/nodejs/index.htm

https://reactjs.org/tutorial/tutorial.html

https://redux.js.org/basics/basic-tutorial/

https://www.tutorialspoint.com/mongodb/index.htm