改訂履歴

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **フロントエンド開発規約** | | **業務名** | | **作成者** | **作成日** |
| 互助事業システム | | 屠 | 2024/7/25 |
| **（改訂履歴）** | | **文書番号** | | **最終更新者** | **最終更新日** |
| GJS-フロントエンド開発規約 | | 屠 | 2024/8/30 |
|  |  |  |  |  |  |
| **版** | **更新日** | **改訂箇所** | **変更内容** | **更新者** | **備考** |
| 1.0 | 2024/7/25 | - | 新規作成 | 屠 |  |
| 1.0 | 2024/8/30 | - | 業務名は「互助防疫システム」から「互助事業システム」に修正する。 | 屠 |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

フロントエンドVUE開発規約

目次

**内容**

[1. コーディング規約 3](#_Toc127369041)

[1.1 命名規則 3](#_Toc127369042)

[1.1.1 項目の命名 3](#_Toc127369043)

[1.1.2 フォルダの命名 3](#_Toc127369044)

[1.1.3 JS、CSS、SCSS、HTML、PNG ファイルの命名 3](#_Toc127369045)

[1.1.4 命名の精確性 4](#_Toc127369046)

[1.2 HTML 規約（Vue Templateにも適用） 4](#_Toc127369047)

[1.2.1 HTMLの型 4](#_Toc127369048)

[1.2.2 インデント 4](#_Toc127369049)

[1.2.3 ブロック注釈 5](#_Toc127369050)

[1.2.4 セマンティックタグ 5](#_Toc127369051)

[1.2.5 コーテーション 5](#_Toc127369052)

[1.3 CSS 規約 5](#_Toc127369053)

[1.3.1 命名 5](#_Toc127369054)

[1.3.2 セレクター 6](#_Toc127369055)

[1.3.3 一括指定プロパティ 6](#_Toc127369056)

[1.3.4 セレクターを一行に 7](#_Toc127369057)

[1.3.5 「0」の後ろ単位を省略 7](#_Toc127369058)

[1.3.6 IDセレクターおよびグローバルタグを避ける 8](#_Toc127369059)

[1.4 LESS 規約 8](#_Toc127369060)

[1.4.1 コーディング 8](#_Toc127369061)

[1.4.2 多重ネストを避ける 9](#_Toc127369062)

[1.5 Javascript/TypeScript 規約 9](#_Toc127369063)

[1.5.1 命名 9](#_Toc127369064)

[1.5.2 コーディングフォーマット 11](#_Toc127369065)

[1.5.3 文字列 12](#_Toc127369066)

[1.5.4 オブジェクトの宣言 12](#_Toc127369067)

[1.5.5 ES6+の使用 12](#_Toc127369068)

[1.5.6 括弧 13](#_Toc127369069)

[1.5.7 undefined の判断 13](#_Toc127369070)

[1.5.8 条件判断のネストは３層まで 13](#_Toc127369071)

[1.5.9 this - self 13](#_Toc127369072)

[1.5.10 console.logの注意事項 13](#_Toc127369073)

[2. Vue 項目規則 14](#_Toc127369074)

[2.1 Vueコーディング基礎 14](#_Toc127369075)

[2.1.1 コンポーネント規約 14](#_Toc127369076)

[2.1.2 テンプレートに簡単な式を使用 17](#_Toc127369077)

[2.1.3 コマンドは略称にする 18](#_Toc127369078)

[2.1.4 タグの順番を一致する 19](#_Toc127369079)

[2.1.5 v-for に key を使用する 19](#_Toc127369080)

[2.1.6 v-show と v-if の使い分け 19](#_Toc127369081)

[2.1.7 script タグの内部順序 19](#_Toc127369082)

[2.1.8 Vue Router 規約 19](#_Toc127369083)

[2.2 Vue プロジェクトフォルダ規約 23](#_Toc127369084)

[2.2.1 基礎 23](#_Toc127369085)

[2.2.2 Vue-cli 足場を使用 23](#_Toc127369086)

[2.2.3 フォルダ説明 23](#_Toc127369087)

[2.2.4 注釈説明 26](#_Toc127369088)

[2.2.5 その他 27](#_Toc127369089)

フロントエンド開発規約

# コーディング規約

## 命名規則

### 項目の命名

すべて小文字で、間にハイフン区切りとする

正解：mall-management-system

誤り： mall-management- system/mallMangementSystem

### フォルダの命名

すべて小文字で、ハイフン区切りにする。複数の場合、後方に「ｓ」を付ける、略称の場合は要らない。 mallManagementSystem

|  |
| --- |
| scripts / styles / components / images / utils / layouts / demo-styles / demo-scripts / img / doc |

正解：

|  |
| --- |
| script / style / demo\_scripts / demoStyles / imgs / docs |

誤り：

【特例】 VUE項目に components 中のフォルダはPascalで命名する。

|  |
| --- |
| HeadSearch / PageLoading |

正解：

|  |
| --- |
| head-search / page-loading / authorized / notice-icon |

誤り：

【特例】VUE 項目 componentsフォルダ以外はkebab-case で命名する。

|  |
| --- |
| page-one / shopping-car / user-management |

正解：

|  |
| --- |
| ShoppingCar / UserManagement |

誤り：

### JS、CSS、SCSS、HTML、PNG ファイルの命名

すべて小文字でハイフン区切りにする。

|  |
| --- |
| render-dom.js / signup.css / index.html / company-logo.png |

|  |
| --- |
| renderDom.js / UserManagement.html |

正解：誤り：

### 命名の精確性

ソースコードにすべて英字で命名する。日本語は禁止する。業務的な項目名はテーブル定義書通り、ローマ字を使用する。業務以外のシステム用語は英語を使用できる

|  |
| --- |
| nendo / list /grid |

正解：

|  |
| --- |
| int 件数 = 3 |

誤り：

## HTML 規約（Vue Templateにも適用）

### HTMLの型

HTML LIVING STANDARDの規則を使用する。text/html フォーマットの HTMLを使用する。XHTMLを避ける。

* エンコードの宣言
* IE 互換性は不要
* charsetの宣言
* doctype は大文字にする

|  |
| --- |
| <!DOCTYPE html> <html>  <head>  <meta charset="UTF-8" />  <title>Page title</title>  </head>  <body>  <img src="images/company-logo.png" alt="Company"> </body>  </html> |

正解：

### インデント

インデントは 2個のスペース（１個 tab）とする。

### ブロック注釈

ブロックエレメント、リストエレメント、テーブルエレメントの前に、HTML注釈をつける必要がある。

### セマンティックタグ

HTML LIVING STANDARD 中のセマンティックタグを優先に使用する。ページの中、 div や pだけの使用をやめよう。

正解

<header></header>

<footer></footer>

誤り

|  |
| --- |
| <div>  <p></p>  </div> |

### コーテーション

シングルコーテーション(’ ') ではなく、ダブルコーテーション(" ") を使用する。

|  |
| --- |
| <div class=”box”></div> |

正解

|  |
| --- |
| <div class=’box’></div> |

誤り

## CSS 規約

### 命名

* クラス名は小文字で記述、間にハイフンで区切りにする。
* id はキャメルケースにする。
* scss 中の変数、関数、placeholderはキャメルケースで命名する。

ID とclassの名前はエレメントの用途を示す言葉または汎用的な名詞で命名しよう。

正解:

.heavy {

font-weight: 800;

}

.important {

color: red;

}

良くない：

|  |
| --- |
| .fw-800 { font-weight: 800;  }  .red { color: red;  } |

### セレクター

1) css のセレクターはタグ名を避ける

できるだけ css の中、HTML タグを避ける。Cssセレクターにタグ名を使用すると、潜在問題の可能性があり、使用しないでください。

2) 子孫セレクターの使用

セレクターチェックを書くとき、直接の子セレクターを使用しないプログラマーが多いだろう。（直接の子セレクターと子孫セレクターの区別に気を付けて）。時には、性能問題になる。汎用的な処理で、DOM 末端のセレクターでない限り、通常、直接の子セレクターを使用する。

悪い例:

.content .title { font-size: 2rem;

}

良い例:

.content > .title { font-size: 2rem;

}

### 一括指定プロパティ

1. プロパティの省略

悪い例：

border-top-style: none; font-family: palatino, georgia, serif; font-size: 100%; line-height: 1.6; padding-bottom: 2em; padding-left: 1em; padding-right: 1em; padding-top: 0;

良い例：

border-top: 0;

font: 100%/1.6 palatino, georgia, serif; padding: 0 1em 2em;

1. プロパティの順序

一括指定プロパティは、置き換えるプロパティの指定順序をなるべく強制しないようにする。いくつかのプロパティが同じ値を持つことができるときは分類して記述する。

### セレクターを一行に

悪い例：

|  |
| --- |
| button {  width: 100px; height: 50px; color: #fff;  background: #00a0e9;  } |

良い例：

button {

width: 100px;

height: 50px;

color: #fff;

background: #00a0e9;

}

### 「0」の後ろ単位を省略

悪い例：

div {

padding-bottom: 0px; margin: 0em;

}

良い例：

div {

padding-bottom: 0; margin: 0;

}

### IDセレクターおよびグローバルタグを避ける

全体のスタイルを影響しないように、IDセレクターおよびグローバルタグをできるだけ使用しない。

悪い例：

#header { padding-bottom: 0px; margin: 0em;

}

良い例：

.header { padding-bottom: 0px; margin: 0em;

}

## LESS 規約

### コーディング

1) 共通 lessファイルを共通cssフォルダーに置く

（ 例：style/less/common ）

例: // color.less、common.less

2) 次の順番で

①@import;

②変数宣言;

③スタイル宣言;

|  |
| --- |
| @import "mixins/size.less";  @default-text-color: #333;  .page {  width: 960px; margin: 0 auto;  } |

### 多重ネストを避ける

細かすぎるCSSセレクターを避け、可読性を保証するために、ネストの深さは3重までにしよう。4重以上のネストは見直しよう。

悪い例：

|  |
| --- |
| .main {  .title {  .name {  color: #fff;  }  }  } |

良い例：

|  |
| --- |
| .main-title {  .name {  color: #fff;  }  } |

## Javascript/TypeScript 規約

### 命名

1) 命名はローワーキャメルケースにする。項目名以外、アンダーバーを使用しない。「＄」記号で終了しない。

|  |
| --- |
| \_name / name\_ / name$ |

誤り：

2) メソッド名、パラメータ名、メンバー変数、ローカル変数はすべて ローワーキャメルケース にする

例：

|  |
| --- |
| localValue / getHttpMessage() / inputUserId |

正解：

|  |
| --- |
| saveShopCarData /openShopCarInfoDialog |

メソッド名は動詞また動詞＋名詞にする

正解：

|  |
| --- |
| save / open / show / go |

誤り：

新規、更新、削除、検索の場合、以下の 5 単語を使用する。

add / update / delete / detail / get

ペアになっている単語：

inject 挿入/extract 取り込み

attach アタッチ/detach ディタッチ

bind バインド/separate 分離

view 参照/browse 閲覧

edit 編集/modify 修正

select 選択/mark マーク

copy コピー/paste ペースト

undo 撤回/redo 再実行

insert 挿入/delete 削除

add 追加/append アペンド

clean クリーン/clear クリア

index 索引/sort ソート

find 探し/search 検索

increase 増加/decrease 減少

play プレー/pause 停止

launch 別プロセス実行/run 実行

compile コンパイル/execute 実行

debug ディバグ/traceトレース

observe 観察/listen 監視

build ビルド/publish 公開

input インポート/output アウトプット

encode エンコード/decode デコード

encrypt 暗号化/decrypt 解読

compress 圧縮/decompress 解凍

parse パース/emit 生成

connect 接続/disconnect 切断

send 送信/receive 受信

download ダウンロード/upload アップロード

refresh 更新/synchronize 同期

update 更新/revert 復旧

lock ロック/unlock 解除

submit サブミッド/commit コミット

push プッシュ/pullプール

expand 展開/collapse 折り畳み

enter 進入/exit 退出

abort 放棄/quit 離れる

collect 収集/aggregate 集める

get 取得/set セット

add 追加/remove 削除

create 新規/destory 消滅

start 開始/stop 停止

open 開く/close 閉じる

read 読み込み/write 書き込み

load ロード/save 保存

begin 開始/end 終了

backup バックアップ/restore復元

import インポート/export エクスポート

split 分割/merge 結合

3) 定数はすべて大文字に、単語間はアンダーバーにする。長くしても良いが、正確な表現にしよう。

正解： MAX\_YOYAKU\_COUNT

|  |
| --- |
| MAX\_COUNT |

誤り：

4) インターフェースクラスはテーブル定義の命名規則にする

インターフェースクラスはバックエンドと統一して、項目名は小文字のローマ字にする。

### コーディングフォーマット

1) インデントは ２個のスペース

正解：

if(x < y){

x += 10;

}

else{

x += 1

}

2) 段落の間に、１行の空行を

読みやすくために、違う処理、ロジック、業務の間、空行を追加する。

### 文字列

シングルコーテーション(‘)を使用、ダブルコーテーション(“)を使用しない。

（こうすることで、HTML文字列を作成しやすい）

正解:

let str = 'foo';

let testDiv = '<div id="test"></div>';

誤り:

let str = 'foo'; let testDiv = "<div id='test'></div>";

### オブジェクトの宣言

1) リテラル値を使用してオブジェクトを定義する

|  |
| --- |
| let user = {}; |

正解：

|  |
| --- |
| let user = new Object(); |

誤り：

2) リテラル値を使って、コンストラクターを定義

正解：

var user = {

age: 0,

name: 1,

city: 3

};

誤り：

var user = new Object(); user.age = 0; user.name = 0; user.city = 0;

### ES6+の使用

ES6+ に提供する関数と文法を優先に使用する。

例えば矢印関数、await/async 、 let 、 for…of など。

### 括弧

以下のキーワードの後ろ、１行だけでも「{」は必要とする：if、 else、 for、 while、 do、 switch、 try、 catch、finally、 with。

正解：

if (condition) {

doSomething();

}

誤り：

if (condition) doSomething();

### undefined の判断

undefinedを使って、変数の判断を禁止する。 typeof を使ってください。

正解：

|  |
| --- |
| if (typeof person === 'undefined') {  ...  } |

誤り：

|  |
| --- |
| if (person === undefined) {  ...  } |

### 条件判断のネストは３層まで

単純比較の場合、三項演算子を使える場合、優先に使用する。ただし、長すぎる三項演算子を避けよう。条件分岐の場合、ネストが多ければ、注釈をつける。３層以上の場合、関数を作成する。

### this - self

同プログラムに thisを参照は ’self’を使用。

### console.logの注意事項

console.log を大量に使用すると、性能に影響するので、使用後削除してください。

# Vue 項目規則

## Vueコーディング基礎

vue 項目規約はVueのオフィシャルサイト（https://v2.cn.vuejs.org/v2/style-guide/）の A規約をベースにする。

### コンポーネント規約

1) コンポーネントは複数単語

HTML エレメントの名前を競合しないため、コンポーネントは複数単語を使用する。

命名はpascal-case に従う。

正解：

|  |
| --- |
| export default { name: 'TodoItem'  // ...  }; |

誤り：

|  |
| --- |
| export default { name: 'Todo',  // ...  }  export default { name: 'todo-item',  // ...  } |

2) ファイル名は kebab-case 形式

正解：

components/|my-component.vue

誤り：

components/ |MyComponent.vue

|- myComponent.vue

3) ベースファイル名は先頭に base を付ける、単語は略称をしない。

正解：

components/

|- base-button.vue

|- base-table.vue

|- base-icon.vue

誤り：

components/ |- myButton.vue

|- VueTable.vue

|- Icon.vue

4) 親コンポーネントに関連する子コンポーネントは親コンポーネント名を前にする。

正解：

components/

|- todo-list.vue

|- todo-list-item.vue

|- todo-list-item-button.vue

|- user-profile-options.vue （完全な単語）

誤り：

components/ |- TodoList.vue

|- TodoItem.vue

|- TodoButton.vue

|- UProfOpts.vue （略称）

5) Template中のコンポーネントは kebab-case 形式で定義する、自己閉じるタグを使用。

正解：

<!—単一ファイルコンポーネント、文字列テンプレート JSXの中 -->

正解：

<my-component />

<Row><table :column="data"/></Row>

誤り：

<MyComponent /> <row><table :column="data"/></row>

6) コンポーネントの data は関数にしなければならない

コンポーネントの中data プロパティ、 new Vue 以外の場合、オブジェクト関数で返さなければならない。もし、オブジェクトを直接返せば、子コンポーネントの間に相互影響が発生する。

正解：

|  |
| --- |
| export default { data () { return { name: 'jack'  }  }  } |

誤り：

|  |
| --- |
| export default { data: { name: 'jack'  }  } |

7) Propはできるだけ詳細に定義

* キャメルケースで命名
* 型を指定する
* 注釈は必要
* required またdefaultを付ける
* 必要な場合、validator を付ける

正解：

|  |
| --- |
| props: {  // コンポーネントの状態、色を制御する  status: { type: String, required: true,  validator: function (value) { return [  'succ',  'info',  'error'  ].indexOf(value) !== -1  }  },  // ユーザレベル、表示個数を制御するため  userLevel：{ type: String,  required: true  } |

8) コンポ-ネントのスタイルにスコープを設定する

正解：

|  |
| --- |
| <template>  <button class="btn btn-close">X</button>  </template>  <!-- `scoped` を設定 -->  <style scoped>  .btn-close { background-color: red;  }  </style> |

誤り：

|  |
| --- |
| <template>  <button class="btn btn-close">X</button>  </template>  <!-- `scoped` 以外の場合 -->  <style>  .btn-close { background-color: red;  }  </style> |

9) プロパティが多い場合、改行をする

正解：

<my-component foo="a" bar="b" baz="c" foo="a" bar="b" baz="c" foo="a" bar="b" baz="c"

/>

誤り：

< my-component foo="a" bar="b" baz="c" foo="a" bar="b" baz="c" foo="a" bar="b" baz="c" foo="a" bar="b" baz="c"/>

### テンプレートに簡単な式を使用

コンポーネントテンプレートに簡単な式しか使用しない。複雑な式は計算のプロパティかメソッドに使用する。計算方法ではなく、その結果を表現するべきである。また、計算のプロパティとメソッドは再利用するメリットがある。

正解：

|  |
| --- |
| <template>  <p>{{ normalizedFullName }}</p>  </template>  // 複雑の式はプロパティを利用  computed: { normalizedFullName: function () { return this.fullName.split(' ').map(function (word) { return word[0].toUpperCase() + word.slice(1)  }).join(' ')  }  } |

誤り：

|  |
| --- |
| <template>  <p>  {{  fullName.split(' ').map(function (word) { return word[0].toUpperCase() + word.slice(1)  }).join(' ')  }}  </p>  </template> |

### コマンドは略称にする

コマンドは略称方法を使用、( :を使用して v-bind: を表示し、 @ を使って v-on:を表示し、 #を使て v-slot:を表示する)

正解：

|  |
| --- |
| <input  @input="onInput"  @focus="onFocus"  > |

誤り：

<input

v-on:input="onInput"

@focus="onFocus"

>

### タグの順番を一致する

同じファイルにコンポーネントは名前昇順にする。

正解：

<template>...</template>

<script>...</script>

<style>...</style>

誤り：

<template>...</template>

<style>...</style>

<script>...</script>

### v-for に key を使用する

### v-show と v-if の使い分け

実行するとき、頻繁に切り替える場合、v-showを使用する。反対に、あまり変わらない場合、v-ifを使用する。

### script タグの内部順序

name > components > mixins > props > data > computed > watch > filter > ホック関数の実行順番は[**beforeCreate、 created**,**beforeMount、 mounted、 beforeUpdate、 beforeUnmount、 unmounted、 beforeDestroy、 destroyed**]） > methods

### Vue Router 規約

1) ページ間の遷移とパラメータの引き渡し

ページの遷移、例えば A画面からB画面、A画面からパラメータをB画面に渡す必要がある。Vuexに一時保存ではなく、ルーターのパラメータを使用する。Vuexの方法を使用する場合、 B画面更新の時、vuexデータが消えて、正常に表示できないなる恐れがある。

正解：

let id = ' 123';

this.$router.push({ name: 'userCenter', query: { id: id } });

2) 遅延ローディング

|  |
| --- |
| {  path: '/uploadAttachment', name: 'uploadAttachment', meta: { title: 'アップロード'  },  component: () => import('@/view/components/uploadAttachment/index.vue')  }, |

3) routerの 命名規約

path、childrenPoints の命名規則は kebab-caseに準ずる（ vue ファイルのフォルダ構造の規約と一致するためである。フォルダ名、ファイルを kebab-caseにすることで、見つけやすいからである） name 命名を kebab-case にすることで component 部品名に一致することができる。（keep-alive 属性を維持するとき、keep-alive はcomponent の name を使って保存するため、名前を一致しなければならない）

|  |
| --- |
| // ダイナミックローディング  export const reload =  [{ path: '/reload',  name: 'reload',  component: Main,  meta: { title: 'ダイナミックローディング',  icon: 'icon iconfont'  },  children: [{  path: '/reload/smart-reload-list',  name: 'SmartReloadList',  meta: { title: 'SmartReload', childrenPoints: [{ title: '検索', name: 'smart-reload-search'  },  {  title: '実行 reload', name: 'smart-reload-update'  },  {  title: '実行結果をチェック',  name: 'smart-reload-result'  }]  },  component: () = >import('@/views/reload/smart-reload/smart-reload-list.vue')  }]  }]; |

4) router 中の path 命名規約

path はkebab-caseの規約を使う以外に、必ず 「/」で始める、children 中のpathでも 同様。

目的：

こんな場合：あるページに問題が発生するとき、速やかにそのvue ファイルを見つかる必要、/を始めに使わないなら、path は parent と children で作られて、router ファイルの中で数回検索する可能性がある。/を始めに使うなら、より速く結果を見つける。

|  |
| --- |
| { path: '/file', name:  'File', component:  Main, meta: { title:  'ファイルサービス  ',  icon: 'ios-cloud-upload'  },  children: [{ path: '/file/file-list', name: 'FileList',  component: () = >import('@/views/file/file-  list.vue')  },  { path: '/file/file-add', name: 'FileAdd', component:  () = >import('@/views/file/file-add.vue')  },  { path: '/file/file-update',  name: 'FileUpdate', component: () = >import('@/views/file/file-update.vue')  }]  } |

4) router 中の path 命名規約

path はkebab-caseの規約を使う以外に、必ず 「/」で始める、children 中のpathでも 同様。

|  |
| --- |
| { path: '/file', name: 'File', component: Main, meta: { title: 'ファイルサービス',  icon: 'ios-cloud-upload'  },  children: [{ path: '/file/file-list', name: 'FileList', component: () = >import('@/views/file/file-list.vue')  },  { path: '/file/file-add', name: 'FileAdd', component: () = >import('@/views/file/file-add.vue')  },  { path: '/file/file-update',  name: 'FileUpdate', component: () = >import('@/views/file/file-update.vue')  }]  } |

目的：

こんな場合：あるページに問題が発生するとき、速やかにそのvue ファイルを見つかる必要、/を始めに使わないなら、path は parent と children で作られて、router ファイルの中で数回検索する可能性がある。/を始めに使うなら、より速く結果を見つける。

## Vue プロジェクトフォルダ規約

### 基礎

vue プロジェクトの中、すべての命名はバックエンドのと統一する。

権限例：バックエンド privilege、 フロントエンドのrouter 、 store、 api など必ずprivielege 単語を使う。

### Vue-cli 足場を使用

vue-cli3 を使用し、プロジェクトを初期化させ、プロジェクト名は上記命名規則を参照。

### フォルダ説明

フォルダ名は上記により、その中の components コンポーネントはキャメル記法、その他components コンポーネントフォルダを除き、すべてのフォルダの命名はkebab-case となる。

|  |  |  |
| --- | --- | --- |
| src | | ソースコードリスト |
| |-- api | | すべて api インタフェース |
| |-- assets | | 静的リソース：images、 icons、 styles など |
| |-- components | | 共通コンポーネント |
| |-- config | | コンフィグファイル |
| |-- constants | | 定数情報、プロジェクトすべての Enum、グローバル定数など |
| |-- directives | | カスタム命令 |
| |-- filters | | フィルター、グローバルツール |
| |-- datas | | シミュレーションデータ、一時保存 |
| |-- lib | | 外部引用のブラグイン保存とファイル変更 |
| |-- mock | | アナログインタフェース、一時保存 |
| |-- plugins | | ブラグイン、すべて使用 |
| |-- router | | ルーター、統一管理 |
| |-- store | | Vuex、統一管理 |
| |-- themes | | CSS |
| |-- views | | Viewsフォルダ |
| | |-- role | | role モジュール名 |
| | |-- |-- role-list.vue | | role テーブル |
| | | |-- |-- role-add.vue | role テーブル新規作成 |
| | | |-- |-- role-update.vue | role テーブル更新 |
| | | |-- |-- index.less | role モジュールスタイル |
| | | |-- |-- components | role モジュール共通コンポーネントファイル |
| | | |-- employee | employee モジュール |

1) api フォルダ

* ファイル、変数ノミネートはバックエンドと一致する必要
* このフォルダはバックエンドのAPI インタフェースに対応、バックエンドによる一つのcontroller に一つの api.js ファイル。プロジェクトが大きい場合、業務によるサブディレクトリを分け、バックエンドと一致する。
* api 中の方法名はバックエンドのapi urlと の意義素の一致性を保つ。
* api 中の方法ごとに注釈を加え、注釈はバックエンドのswagger ファイルと一致する。

例：

|  |
| --- |
| バックエンド url： EmployeeController.java  /employee/add  /employee/delete/{id}  /employee/update フロントエンド： employee.js  // 従業員増加  function addEmployee(data) {  return postAxios('/employee/add', data)  }  // 従業員情報更新  function updateEmployee(data) {  return postAxios('/employee/update', data)  }  // 従業員削除  function deleteEmployee(employeeId) {  return postAxios('/employee/delete/' + employeeId)  }  // 従業員の情報  function getEmployee(data) {  return postAxios('/employee/get/', data)  }  // 従業員の情報リスト  function getEmployeeList(data) {  return postAxios('/employee/getlist/', data)  } |

2) assets フォルダ

assets は静的リソース、その中にimages,、styles、 iconsなどの静的リソースを保存され、 静的リソースノミネートフォームはkebab-caseとなる

|assets

|-- icons

|-- images

| |-- background-color.png

| |-- upload-header.png

|-- styles

3) components フォルダ

このフォルダはコンポーネントによるフォルダを分け、フォルダ名はkebab-case、コンポーネントの命名規則も kebab-caseでよい。

|components

|-- error-log

| |-- index.vue

| |-- index.less

|-- markdown-editor

| |-- index.vue

| |-- index.js |-- kebab-case

4) constants フォルダ

フォルダにプロジェクトのすべての定数を保存する、定数を vue に使うなら、vue-enum プラグインを使用ください。

(https://www.npmjs.com/package/vue-enum)

|  |
| --- |
| **フォルダ**構造：  |constants  |-- index.js  |-- role.js  |-- employee.js  例子： employee.js  export const EMPLOYEE\_STATUS = {  NORMAL: { value: 1, desc: '異常なし'  },  DISABLED: { value: 1, desc: '使用禁止'  },  DELETED: { value: 2,  desc: '削除済'  }  };  export const EMPLOYEE\_ACCOUNT\_TYPE = {  USERNAME: { value: 4,  desc: 'ユーザー暗証番号登録'  }  };  export default {  EMPLOYEE\_STATUS,  EMPLOYEE\_ACCOUNT\_TYPE  }; |

5) router と store コンテンツ

二つのフォルダは必ず業務を見分ける必要で、同じjs ファイルに入れられない。

router はできるだけviewsの中の構造と一致し store は業務ごとに見分けて、異なるjs ファイルになる

6) views リスト

ノミネートはバックエンド、router、apiなど一致にする

components の中、コンポーネントはPascalCase の規則を使用する。

|  |  |  |
| --- | --- | --- |
| |-- views | | Viewsリスト |
| | | |-- role | role モジュール名 |
| | | | |-- role-list.vue | role リストページ |
| | | | |-- role-add.vue | role 新規作成 |
| | | | |-- role-update.vue | role ページを更新 |
| | | | |-- index.less | role モジュールスタイル |
| | | | |-- components | role モジュール共通コンポーネントファイル |
| | | | | |-- role-header.vue | role ヘッドアッセンブリ |
| | | | | |-- role-modal.vue | role ポップアップコンポーネント |
| | | |-- employee | employee モジュール |
| | | |-- behavior-log | 処理ログ |
| | | |-- code-generator | ソースコードジェネレータ |

### 注釈説明

注釈を加える必要なところの整理

* 共通コンポーネント使用説明
* api ファイルのインタフェース js ファイルに注釈を加える
* store 中のstate、mutation、 action などに注釈を加える。
* vue ファイルのtemplate に注釈を加える、データ大きな場合 start end に注釈を加える。
* vue ファイルのmethods、 methodごとに 注釈を加える
* vue ファイルのdata、 珍しい単語に注釈を加える

### その他

vue の利用なので、開発の中 vue のデータ駆動でDOMを更新する、 DOMの手動操作禁止、禁止内容： dom の増加と削除、スタイル変更、イベント追加など。

1) デッドコード削除

git/svn などコードツールの利用で、デッドコードを即時削除する必要、例： console 構文のデバッグ、デッドコード。

Vue3 糖衣構文のプログラムを優先採用：

Vue3 script構造と注釈例

name > components > mixins > props > data(ref…) > computed > watch > filter > フック関数（フック関数は順番により実行[**beforeCreate、 created**、**beforeMount、 mounted、 beforeUpdate、 beforeUnmount、 unmounted、 beforeDestroy、 destroyed**]） > methods

注釈例：

//---------------------------------------------------------------------------

//業務名(name > components > mixins)

//---------------------------------------------------------------------------

//---------------------------------------------------------------------------

//属性(props)

//---------------------------------------------------------------------------

//---------------------------------------------------------------------------

//データ定義(data(ref…))

//--------------------------------------------------------------------------

//---------------------------------------------------------------------------

//計算定義(computed)

//--------------------------------------------------------------------------

//---------------------------------------------------------------------------

//監視定義(watch)

//--------------------------------------------------------------------------

//---------------------------------------------------------------------------

//フック関数

//--------------------------------------------------------------------------

//---------------------------------------------------------------------------

//メソッド(methods)

//--------------------------------------------------------------------------