CAS フロントエンド開発ガイド

小川岳史 · 島内鉄矢 · 山﨑大

Table of Contents

このドキュメントについて

このドキュメントの目的

本書ではプロジェクトにおけるフロントエンド開発における詳細設計および実装方法に関する指針を示します。基本設計担当者や実装担当者が、高い一貫性と論理性を備えた設計、実装をできるようにすることが本書の目的です。

このドキュメントから得られるメリット

- 実装の統一性を高める
  - 本書の内容に準じて設計を行うことで、実装の個人差を抑えることができ、システム全体で統一性の高いものにできます
- 設計の考慮抜けを抑える
  - 設計者に対して、設計を行う上で考慮しておかなければならない留意点を喚起することで、考慮漏れを抑えられます

このドキュメントの対象読者

プロジェクトの基本設計担当者および実装担当者を読者として想定しています。
次のことについて理解していることを前提としています。

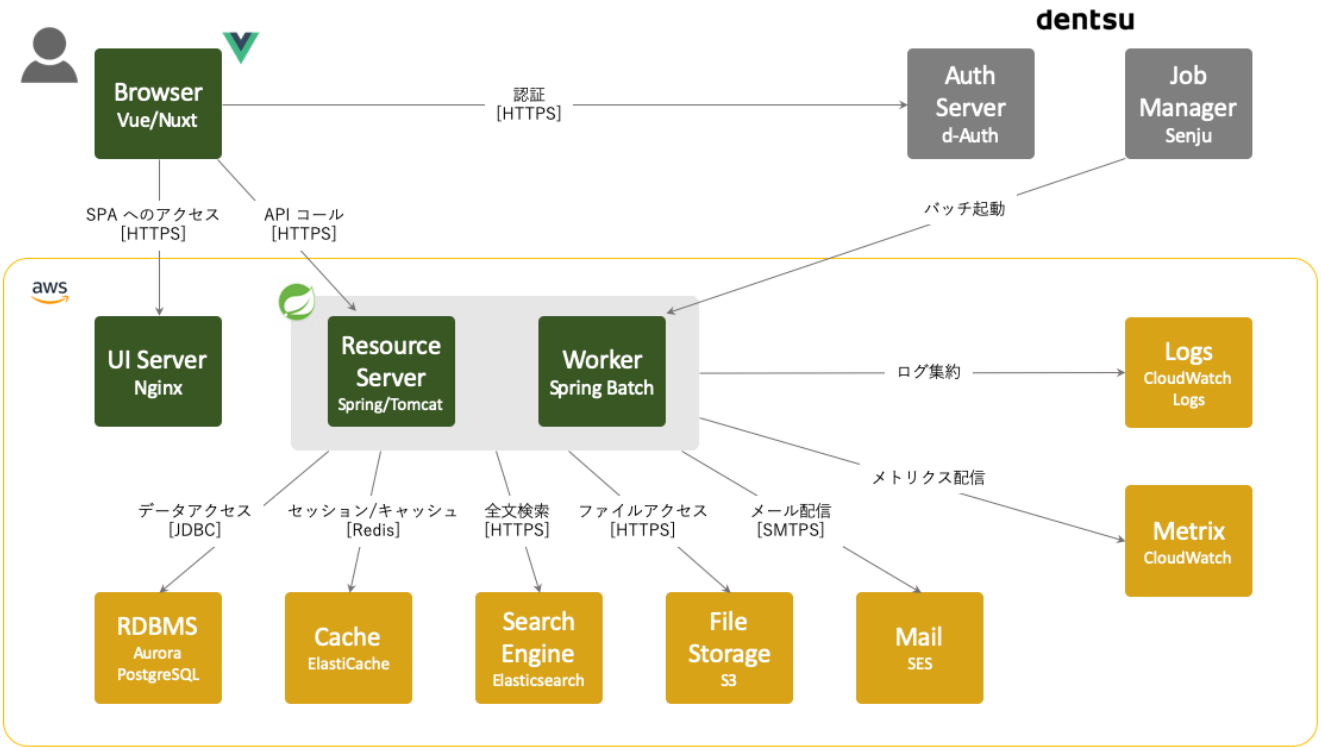
1. Vue.js ガイド (https://jp.vuejs.org/v2/guide/index.html) を読み理解している
2. Vuex ガイド (https://vuex.vuejs.org/ja/) を読み理解している
3. Nuxt.js ガイド (https://ja.nuxtjs.org/guide/) を読み理解している

このドキュメントの書式

本書は Asciidoctor (https://asciidoctor.org/) によって作成しています。

アーキテクチャ

Nuxt.js の SPA モード (https://ja.nuxtjs.org/api/configuration-mode/) をベースとします。



コンポーネント

コンポーネントの設計

コンポーネントはその責務によって Container Component と Presentational Component に別けて設計します。

Presentational と Container Component

	Presentational Components	**Container Components**
目的	見た目（マークアップ, スタイル）	動作（データの取得、状態の更新）
Store へのアクセス	-	✓
データの読込	Props	Subscribe to Redux state
データの変更	Props から取得した Callback を呼ぶ	Vuex の Action を Dispatch する

コンポーネントの粒度

Atomic Design

Category	**Directory**	**State**	**Store**
Atoms	src/components/atoms	-	-
Molecules	src/components/molecules	✓	-
Organisms	src/components/organisms	✓	-
Pages	src/pages	✓	✓
Templates	src/layouts	✓	✓

状態を持たないコンポーネントは基本的に `Functional Component` として定義します。

```vue
<template functional>
```

Storybook

> NOTE
>
> **TODO**
> 本プロジェクトにおける Storybook の使いかた、または参考リンクを追加する

スタイルシート

Scoped CSS および SCSS 記法を使用します。

VUE

```
<template>
  <button class="button button-close">X</button>
</template>

<style lang="scss" scoped>
.button {
  border: none;
  border-radius: 2px;
}

.button-close {
  background-color: red;
}
</style>
```

ルーティング

Nuxt.js のルーティングサポート (https://ja.nuxtjs.org/guide/routing/) の使用を基本とします。

URL 設計

URL はリソース名、一意の識別子、アクション名を使用して組み立てます。

画面	URL	備考
一覧・検索	/resources?q=sample&page=1&size=10&sort=name	検索条件やページング、ソートはクエリーパラメータを使用します
詳細	/resources/:id	:id は一意の識別子を使用します
新規登録	/resources/:id/new	アクション名として new を使用します
編集	/resources/:id/edit	アクション名として edit を使用します
削除	/resources/:id/delete	削除の前の確認画面を準備する場合など
子リソース	/resources/:id/comments	関連するコメント一覧を表示する場合など

ブラウザ履歴の置換

検索条件の変更など画面遷移せずに表示内容を変化させる場合などは利用者が URL をコピーしてその状態を共有できるよう、`$router.replace` の使用を考慮してください。

入力チェック

NOTE

TODO

1. `Element UI` が使用する async-validator (https://github.com/yiminghe/async-validator) を第一候補とし、Vuelidate (https://vuelidate.netlify.com/) or VeeValidate (https://baianat.github.io/vee-validate/) も検討する
2. エラーの表示方法について記述する
   フロントでの入力エラー、サーバーからのエラーなどケース毎に示す

Store

状態の分類

アプリケーションで管理する状態を次のように分類します。

`Domain data`：API から取得したデータ
`App state`：アプリケーションの動作に固有のデータ（"Todo＃5が現在選択されています"、または "Todosを取得する要求が進行中です"など）
`UI state`：UI が現在どのように表示されているかを表すデータ（"EditTodoモーダルダイアログが現在開いている" など）

Store で管理する状態

Store で管理する状態は前述の `Domain data` と `App state` とし、`UI state` は Component で管理します。

モジュール分割

Store は以下のようにモジュール分割します。

```
├── domain ...Domain data
|    ├──── clients ...ドメイン名
|    ├──── sales
|    ├──── purchases
├── app ...App state
|    ├──── user ...ログインしているユーザ情報など
|    ├──── cart ...買い物かごの状態など
```

認証（d-Auth）

なし（ガイドとしては不要・シーケンスを別途）

環境変数

NOTE

TODO
開発、テスト、本番環境等の実行環境に対応した環境変数の書き方を記述する

HATEOAS

NOTE

TODO
サーバーから返却される `HATEOAS` 形式のレスポンスの使いかたを記述する
（リソースの操作に関するリンクの処理方法）

エラーハンドリング

NOTE

TODO
下記エラー処理の使い分けと代表的な使い方を記述

errorCaptured

TODO

nuxt error page

TODO

nuxt error function

TODO

async/await

try-catch or await-catch

TODO

ユニットテスト

> NOTE
>
> **TODO**
> `Store` 、 `Container Component` に対応するテストについて記述する
> UI(見た目)に関わるユニットテストはコストが高いため記載しない予定

Consumer Driven Contact

> NOTE
>
> **TODO**
> Pact を含む、以下 CDC の章を追加する
> 本来の `CDC` の実践は開発初期から導入し、開発を円滑に進めるためのものだが、+ 本プロジェクトにおいては保守フェーズに入り運用を円滑に行うための準備として導入が良いと考えている
> ※ CDC 実践の落とし所については要相談です

Pact

Pact Broker

Todo

Pact に対応したユニットテストの書き方

Todo

コーディング規約

Vue 公式の スタイルガイド (https://jp.vuejs.org/v2/style-guide/index.html) を基本とします。

コンポーネント名

プレフィックスとして `Cas` を付与します。

VUE

```
@Component
export default class CasButton extends Vue {
  // ...
}
```

コンポーネントのファイル名

コンポーネントのファイル名は、すべてケバブケース (kebab-case) とします。

```
components/
|- cas-button.vue
```

コンポーネントの宣言

コンポーネントの宣言は TypeScript デコレータによる `class-style Vue components` を用いることとします。

VUE

```
<template>
  <button class="btn-primary" @click.prevent="handleClick">
    <slot></slot> (clicked - {{ count }})
  </button>
</template>

<script lang="ts">
import { Component, Vue } from 'nuxt-property-decorator'

@Component
export default class CasButton extends Vue {
  count = 0

  handleClick() {
    this.count++
  }
}
</script>

<style lang="scss">
.btn-primary {
  font-size: 1.2rem;
}
</style>
```

デコレータは Nuxt Property Decorator (https://github.com/nuxt-community/nuxt-property-decorator) から import することとします。

VUE

```
import { Component } from 'nuxt-property-decorator'
```

テンプレート内でのコンポーネント名の形式

テンプレート内でのコンポーネント名はすべてケバブケース (kebab-case) とします。

VUE

```
<my-component></my-component>
```

Git ブランチモデル

Git のブランチモデルには GitHub Flow (http://scottchacon.com/2011/08/31/github-flow.html) を使用する。

Git Flow の主なルール

1. master ブランチは常にデプロイ可能な状態にする
2. 作業用ブランチを master から作成する（例：new-oauth2-scopes）
3. 作業用ブランチを定期的にプッシュする
4. プルリクエストを活用する
5. プルリクエストが承認されたら master へマージする
6. master へのマージが完了したら直ちにデプロイを行う

開発ツール

基本的に開発者の使いやすいツールを使用することとします。ただし、ツールの選定において以下の機能がそなわっていることを推奨します。

- デバッグ（ブレイクポイント・ライン実行）
- TypeScript 型推論
- Prettier によるコードフォーマット

Last updated 2019-06-12 08:21:12 UTC