新人エンジニア向け：Vue.jsで学ぶ実務開発のエッセンス

はじめに

実際の保険会社の申込みシステムを例に、Vue.jsを使った実務開発で重要なポイントを学んでいきましょう。単なる技術解説ではなく、「なぜそう設計するのか」という実務の視点を大切にしています。

1. コンポーネント設計の基本思想

再利用可能性を意識したコンポーネント

vue

```
<!-- Stepper.vue - 通常版 -->
<template>
  <li v-bind:class="{ current: step == 2 }">
    <p class="title">契約者<br />情報入力</p>
  </li>
</template>

<!-- StepperKanwa.vue - 緩和型版 -->
<template>
  <li v-bind:class="{ current: step == 2 }">
    <p class="title">告知<br />情報入力</p>
  </li>
</template>
```

学び: 同じUIでも、ビジネス要件が違えば異なるコンポーネントが必要。無理に1つにまとめるより、明確に分けることで保守性が向上します。

プロパティ駆動の設計

javascript

```
export default {
  name: "Stepper",
  props: {
    step: String  // 親から現在のステップを受け取る
  }
};
```

実務のポイント: コンポーネントは「外部から制御される」設計にする。内部で状態を持ちすぎると、予期しない動作の原因になります。

2. 動的スタイリングの実践

条件付きクラスの適用

```
html
```

```
<li v-bind:class="{ current: step == 1 }">
```

これは以下と同じ意味です：

```
javascript
```

```
// stepが1の場合、'current'クラスが追加される
if (step == 1) {
  クラス = 'current'
} else {
  クラス = ''
}
```

実務での応用例:

- 現在のページをナビゲーションで強調
- フォームのバリデーション状態表示
- ボタンの有効/無効状態

3. レスポンシブデザインの実装戦略

デバイス別表示の制御

```
html
```

```
<p class="title mt10 pc-only">意向確認</p>
<p class="title mt05 sp-only">意向確認</p>
```

実務のポイント:

- CSS Media Queryだけでなく、HTMLレベルでも制御する
- スマートフォンとPCで異なるマージンが必要な場合がある
- デザイナーとの綿密な調整が重要

4. アセット管理のベストプラクティス

Webpackエイリアスの活用

```
html
```

```
<img src="@/assets/common_img/icon_step_01.svg" alt="step1" />
```

なぜ`@`を使うのか:

- 相対パスの`../../../`地獄を避ける
- ファイル移動時の修正箇所を最小化
- チーム開発での統一性確保

実務での注意点:

```html
<!-- 悪い例：相対パス -->
<img src="../../../assets/img/icon.svg" />

<!-- 良い例：エイリアス使用 -->
<img src="@/assets/img/icon.svg" />
```

5. ビジネスロジックの実装

保険商品に応じた動的表示

```javascript
mainInsuranceName: function () {
  if (ProductMainTypeCodes.MEDICAL.includes(this.application.mainTypeCd)) {
    if(this.application.medicalExclusiveProductCd == MedicalExclusiveProduct.RABBIT) {
      return MainTypePetName.RABBIT;
    }
    return MainTypePetName.MEDICAL;
  } else if (this.daoFlg()) {
    return MainTypePetName.DAO;
  }
  // ... 他の条件
}
```

実務で重要なこと:

1. **可読性**: 条件が複雑でも、何をチェックしているかが分かる
2. **拡張性**: 新しい商品タイプが追加されても対応しやすい
3. **保守性**: 定数を使用して、マジックナンバーを避ける

定数管理の重要性

```javascript
import { AgencyCode, ProductMainTypeCodes, MainTypePetName } from "@shared/constants"

// 悪い例
if (this.application.agencyCd == "0002") { ... }

// 良い例
if (this.application.agencyCd == AgencyCode.DIRECT) { ... }
```

6. 状態管理とVuex

ストアからのデータ取得

```javascript
data() {
  return {
    application: this.$store.getters["application/application"],
  };
}
```

Vuexを使う理由:

- コンポーネント間でのデータ共有
- 申込み情報のような重要なデータの一元管理
- デバッグツールでの状態追跡

実務のポイント:

- 全てをVuexに入れる必要はない
- ローカル状態とグローバル状態を適切に使い分ける

7. セキュリティとユーザビリティ

ブラウザバック制御

```javascript
mounted() {
  // ブラウザバック時におけるエラー検知除外対応
  history.pushState(null, null, location.href);
  window.addEventListener('popstate', () => {
    history.go(1);
  });
}
```

なぜこれが必要か:

- 申込み完了後に戻ると、データの整合性が崩れる可能性
- 金融商品では特に重要なセキュリティ要件
- ユーザーの誤操作を防ぐ

8. アクセシビリティの配慮

適切なalt属性

```
html
```

```html
<img src="@/assets/common_img/icon_step_01.svg" alt="step1" />
```

実務での考慮点:

- スクリーンリーダーでの読み上げ
- 画像が読み込めない場合の代替テキスト
- SEOへの影響

セマンティックなHTML構造

```
html
```

```html
<section class="step-page-link">
  <ul class="step-page-link-list">
    <li>...</li>
  </ul>
</section>
```

なぜ重要か:

- HTML本来の意味に沿った構造
- アクセシビリティの向上
- 保守性の向上

9. エラーハンドリングと例外処理

実際のプロジェクトで気をつけること

```javascript
// TODOコメントの活用
// TODO: 商品名未確定
} else if (ProductMainTypeCodes.TATE_SHINDAN.includes(this.application.mainTypeCd)) {
  return "なないろがん保険盾 一時金型"
}
```

実務でのTODO管理:

- 未確定仕様は明確にコメント
- 将来の改修ポイントを記録
- チーム内での情報共有

10. 保守性の高いコードを書くために

命名規則の一貫性

```javascript
// 良い例：目的が明確
clickTopPage() { ... }
mainInsuranceName() { ... }
daoFlg() { ... }

// 悪い例：目的が不明確
doSomething() { ... }
func1() { ... }
```

コメントの適切な使用

```javascript
// セブン可変処理
daoFlg() {
  if(this.application.agencyCd == AgencyCode.WDC &&
     this.application.agencyBranchCd == AgencyBranchCode.DAO) {
    return true;
  }
  return false;
}
```

コメントのベストプラクティス:

- 「何をしているか」より「なぜしているか」を説明
- ビジネスロジックの背景を記載

- 将来の開発者（自分も含む）への配慮

新人エンジニアへのアドバイス

1. 完璧を求めすぎない

最初から全てを理解しようとせず、一つずつ着実に学習しましょう。

2. 実際のコードから学ぶ

技術書だけでなく、実際のプロジェクトコードを読む習慣をつけましょう。

3. 「なぜ」を常に考える

技術選択には必ず理由があります。表面的な実装だけでなく、背景も理解しましょう。

4. チーム開発を意識する

自分だけが理解できるコードではなく、チーム全体が保守できるコードを書きましょう。

5. ユーザーの立場で考える

技術的な実装も大切ですが、最終的にはユーザーの体験が最重要です。

まとめ

Vue.jsの技術的な側面だけでなく、実際のビジネス要件に対応するための設計思想や、チーム開発での注意点を学びました。新人エンジニアの皆さんは、まず動くものを作ることから始めて、徐々に品質や保守性を高めていく姿勢が大切です。

実務では技術的な知識だけでなく、ビジネス理解、チームワーク、そしてユーザーへの配慮が求められます。この記事が、皆さんの成長の一助となれば幸いです。

次のステップ:

- 実際にVue.jsプロジェクトを作ってみる
- Vuexでの状態管理を実装してみる
- アクセシビリティチェックツールを使ってみる
- チーム開発でのコードレビューに参加する