

【新人エンジニア必読】Vue.jsテストコードから学ぶ実務の進め方

～商品リニューアルに伴うテストコード改修の完全解説～

はじめに

こんにちは！Vue.js先生です。今回は、実際の保険見積もりシステムのテストコード改修を通じて、Vue.jsの実務での扱い方を学んでいきましょう。

今回学ぶこと

- Vuex（状態管理）の導入方法
- テスト環境の改善テクニック
- 商品リニューアルに伴うコード変更
- 実務で使えるコードレビューのポイント

今回の変更概要

商品「メディカル礎女性専用」が「メディカル兎」にリニューアルされ、それに伴ってテストコードも大幅に改修されました。

主な変更点

1. **Vuex導入** - 状態管理ライブラリの追加
2. **テスト構造改善** - beforeEach/afterEachの活用
3. **商品名変更** - 女性専用 → 兎
4. **コードメンテナンス** - より保守しやすい構造に

Vue.js基礎知識の復習

Vuexとは？

javascript

// Vuexは Vue.js アプリケーション用の状態管理パターン + ライブラリ
// 複数のコンポーネント間でデータを共有する時に使用

// 従来の方法 (Props/Events)

Parent → Child → GrandChild // データを何度も受け渡し

// Vuex使用時

Store ← Component A, B, C // 中央のストアから直接データ取得

テストの基本構造

```
javascript

describe('テスト対象', () => {
  beforeEach(() => {
    // 各テスト実行前の準備
  });

  it('テストケース', () => {
    // 実際のテスト処理
  });

  afterEach(() => {
    // 各テスト実行後のクリーンアップ
  });
});
```

コード変更の詳細解説

1. import文の変更

```
javascript

// 【変更前】
import { mount, createLocalVue } from '@vue/test-utils' // Vue テストユーティリティ
import G01SimulationPage from '@pages/G01SimulationPage.vue' // テスト対象コンポーネント
import flushPromises from 'flush-promises' // 非同期処理待機用
import api from '@services/api' // API通信用モジュール
import MockAdapter from 'axios-mock-adapter' // HTTP リクエストモック
import Router from 'vue-router' // ルーティング機能

// 【変更後】
import { mount, createLocalVue } from '@vue/test-utils' // Vue テストユーティリティ
import G01SimulationPage from '@pages/G01SimulationPage.vue' // テスト対象コンポーネント
import flushPromises from 'flush-promises' // 非同期処理待機用
import api from '@services/api' // API通信用モジュール
import MockAdapter from 'axios-mock-adapter' // HTTP リクエストモック
import Router from 'vue-router' // ルーティング機能
import Vuex from 'vuex' //  Vuex 状態管理ライブラリ
```

解説：

- `import Vuex from 'vuex'` - Vuex状態管理ライブラリを追加
- 実務では新機能導入時に必要なライブラリを段階的に追加


2. 初期設定の変更

javascript

// 【変更前】

```
const mockAxios = new MockAdapter(api) // APIのモックアダプターを作成
const localVue = createLocalVue()      // テスト専用のVueインスタンスを作成
localVue.use(Router)                   // ルーター機能をテスト環境に登録
```

// 【変更後】

```
const mockAxios = new MockAdapter(api) // APIのモックアダプターを作成
const localVue = createLocalVue()      // テスト専用のVueインスタンスを作成
localVue.use(Router)                   // ルーター機能をテスト環境に登録
localVue.use(Vuex)                     //  Vuex機能をテスト環境に登録
```

解説：

- `createLocalVue()` - テスト専用のVueインスタンスを作成
- `localVue.use()` - プラグインを登録（Router, Vuex等）
- 本番環境に影響せずテスト環境だけでライブラリを使用可能


3. グローバル変数の追加

javascript

// 【変更前】

```
describe('G01SimulationPage.vue', () => {
  // 各テスト内で個別にwrapperを作成
```

// 【変更後】

```
describe('G01SimulationPage.vue', () => {
  let store //  Vuexストアインスタンスを格納する変数
  let wrapper //  マウントしたコンポーネントを格納する変数
```

解説：

- `let store` - Vuexストアインスタンスを格納
- `let wrapper` - マウントしたコンポーネントを格納
- テスト間でインスタンスを再利用し、効率化を図る

4. beforeEach/afterEach の導入

javascript

```
beforeEach(() => {  
  // 各テスト実行前に毎回実行される処理  
  store = new Vuex.Store({  
    // 新しいVuexストアインスタンスを作成  
    // モジュール分割でストアを整理  
    modules: {  
      // agency (代理店) モジュール  
      agency: {  
        // 名前空間を有効化 (重要!)  
        namespaced: true,  
        // データの保存場所  
        state: {  
          // 代理店情報の初期値  
          agency: undefined  
        },  
        // 非同期処理 (API呼び出し等)  
        actions: {  
          getAgency: jest.fn().mockResolvedValue({ // モック関数でAPIをシミュレート  
            medicalRabbitDisplayFlag: 1 // 兎商品表示フラグ  
          })  
        },  
        // stateから値を取得する関数群  
        getters: {  
          agency: state => state.agency // state.agencyをそのまま返すgetter  
        },  
        // stateの値を変更する関数群  
        mutations: {  
          set: (state, payload) => { // データをstateにセット  
            state.agency = payload.agency  
          },  
          clear: state => { // stateをクリア  
            state.agency = undefined  
          }  
        }  
      }  
    }  
  })  
})
```

ステップ解説：

1. **namespaced: true** - モジュールに名前空間を設定

javascript

// 名前空間ありの場合

```
this.$store.dispatch('agency/getAgency')
```

// 名前空間なしの場合

```
this.$store.dispatch('getAgency')
```

2. **state** - データの保存場所

javascript

```
state: { agency: undefined } // agency情報を保存
```

3. **actions** - 非同期処理（API呼び出し等）

javascript

```
getAgency: jest.fn().mockResolvedValue({  
  medicalRabbitDisplayFlag: 1  
})  
// テスト用のモック関数で、Promise成功を返す
```

4. **getters** - stateから値を取得

javascript

```
agency: state => state.agency  
// stateのagency値をそのまま返すgetter
```

5. **mutations** - stateの値を変更

javascript

```
set: (state, payload) => {  
  state.agency = payload.agency;  
}  
// payloadからagency情報をstateに保存
```

5. afterEach の追加

javascript

```
afterEach(() => {  
  // 各テスト実行後のクリーンアップ処理  
  wrapper.vm.$router.push({ query: { parm4: '' } }) // ルートパラメータをクリア  
})
```

解説：

- **afterEach** - 各テスト後に実行される処理
- **wrapper.vm.\$router.push()** - ルートパラメータをクリア
- テスト間でのデータ汚染を防ぐ重要な処理

6. DOM要素名の変更

javascript

// 【変更前】

```
const dummyElements = {  
  sumameAccordionTarget: { style: { height: -1 } }, // 基本商品アコーディオン  
  sumameAccordionBody: { clientHeight: 100 }, // 基本商品本体  
  womansSumameAccordionTarget: { style: { height: -1 } }, // 女性専用アコーディオン  
  womansSumameAccordionBody: { clientHeight: 100 } // 女性専用本体  
  // ...  
}
```

// 【変更後】

```
const dummyElements = {  
  sumameAccordionTarget: { style: { height: -1 } }, // 基本商品アコーディオン  
  sumameAccordionBody: { clientHeight: 100 }, // 基本商品本体  
  sumameRabbitAccordionTarget: { style: { height: -1 } }, //  兎商品アコーディオン  
  sumameRabbitAccordionBody: { clientHeight: 100 } //  兎商品本体  
  // ...  
}
```

解説：

- `womansSumame` → `sumameRabbit` に命名変更
- 商品名変更に伴うHTML要素名の統一
- 実務では商品仕様変更時にコードも同期して変更

7. テストケースの変更

javascript

// 【変更前】

```
it('ケース 12-2 メディカル礎女性専用 アコーディオンがアニメーションすること', async () => {  
  // テスト処理...  
})
```

// 【変更後】

```
it('ケース 12-2 メディカル兎 アコーディオンがアニメーションすること', async () => {  
  // テスト処理...  
})
```

解説：

- テストケース名も商品名に合わせて変更
- 一貫性のあるネーミングが保守性を向上


8. wrapper作成の変更

javascript

// 【変更前】

```
const wrapper = mount(G01SimulationPage, { // コンポーネントをマウント
  localVue, // テスト用Vueインスタンス
  stubs: ['PageTopButton'], // スタブ化するコンポーネント
  mocks: { // モックオブジェクト
    $store, // ストアのモック
    $cookies // クッキーのモック
  },
  router: new Router() // ルーターインスタンス
})
```

// 【変更後】

```
wrapper = mount(G01SimulationPage, { // グローバル変数に代入
  localVue, // テスト用Vueインスタンス
  stubs: ['PageTopButton'], // スタブ化するコンポーネント
  mocks: { // モックオブジェクト
    $cookies // クッキーのモックのみ
  },
  store, //  実際のVuexストアを渡す
  router: new Router() // ルーターインスタンス
})
```

解説：

- `mocks: { $store }` → `store` に変更
- 実際のVuexストアインスタンスを使用
- より本番環境に近いテスト環境を構築

商品変更に伴うUI要素の変更

モーダル表示処理の変更

javascript

// 【変更前】

```
it('ケース 20-8 メディカル礎女性専用 女性専用商品について押下時', async () => {  
  // ...テスト準備...  
  
  const modal = wrapper.find('.planModal-iframe') // モーダル内のiframe要素を取得  
  expect(modal.exists()).toBeTruthy() // モーダルが存在することを確認  
  
  const iframeSrc = modal.attributes('src') // iframe のsrc属性を取得  
  expect(iframeSrc).toBe('\\\\public\\\\womans_product.html') // 女性専用ページを確認  
  
  wrapper.vm.closeModalDetails() // 女性専用モーダルクローズ  
  expect(wrapper.vm.modalDetails.womansSumameProduct).toBeFalsy() // 状態確認  
})
```

// 【変更後】

```
it('ケース 20-8 メディカル兎 兎商品について押下時', async () => {  
  // ...テスト準備...  
  
  const modal = wrapper.find('.planModal-iframe') // モーダル内のiframe要素を取得  
  expect(modal.exists()).toBeTruthy() // モーダルが存在することを確認  
  
  const iframeSrc = modal.attributes('src') // iframe のsrc属性を取得  
  expect(iframeSrc).toBe('\\\\public\\\\rabbit_product.html') // NEW 兎ページを確認  
  
  wrapper.vm.closeModalRabbitDetails() // NEW 兎専用モーダルクローズ  
  expect(wrapper.vm.modalDetails.sumameRabbitProduct).toBeFalsy() // 状態確認  
})
```

ステップ解説：

1. iframeSrc確認

javascript

```
expect(iframeSrc).toBe('\\\\public\\\\rabbit_product.html')
```

- モーダル内のiframeが正しいHTMLファイルを表示
- 商品変更に伴ってHTMLファイル名も変更

2. モーダルクローズ処理

javascript

```
wrapper.vm.closeModalRabbitDetails()
```

- 兎商品専用のクローズメソッドを呼び出し
- 機能ごとに専用メソッドを用意（保守性向上）

3. 状態確認

javascript

```
expect(wrapper.vm.modalDetails.sumameRabbitProduct).toBeFalsy()
```

- モーダルが正しく閉じられたことを確認
- 状態管理オブジェクトの値検証

フッター表示ロジックの変更

javascript

// 【変更前】

// フッタ部の選択中商品文言に「メディカル礎女性専用」が表示されていること

```
expect(wrapper.find('.footer-incurance-block-sumame').exists()).toBe(false) // 基本商品は
```

```
expect(wrapper.find('.footer-incurance-block-womans-sumame').exists()).toBe(true) // 女性専用商
```

// 【変更後】

// フッタ部の選択中商品文言に「メディカル兔」が表示されていること

```
expect(wrapper.find('.footer-incurance-block-sumame').exists()).toBe(false) // 基本商品に
```

```
expect(wrapper.find('.footer-incurance-block-sumame-rabbit').exists()).toBe(true) // 兔商品は
```



解説：

- CSS クラス名も商品名に合わせて変更
- `womans-sumame` → `sumame-rabbit`
- フロントエンドではHTML/CSS/JSが密接に連携

実務で活かせるポイント

1. 段階的なリファクタリング

javascript

//  一度に全て変更

// 全ファイルを同時に変更 → 不具合発生時の原因特定困難

//  段階的に変更

// 1. Vuex導入


// 2. テスト構造改善

// 3. 商品名変更

// 4. 最終動作確認

2. 命名規則の統一

javascript

```
//  統一された命名
sumameRabbitAccordionTarget    // アコーディオン要素
sumameRabbitAccordionBody      // アコーディオン本体
closeModalRabbitDetails        // モーダルクローズ関数
modalDetails.sumameRabbitProduct // 状態管理プロパティ
```

3. テストの独立性確保

javascript

```
// beforeEach - 毎回クリーンな状態でテスト開始
beforeEach(() => {
  store = new Vuex.Store({ /* 初期状態 */ }) // 新しいストア作成
})

// afterEach - テスト後の後処理
afterEach(() => {
  wrapper.vm.$router.push({ query: { parm4: '' } }) // ルート状態クリア
})
```

4. モックとリアルを使い分け

javascript

```
// モック使用 (外部API)
mockAxios.onPost('/date_calculate/').reply(200, getDateCalculate())

// リアル使用 (Vuexストア)
store, // 実際のVuexストアインスタンス
```

Vue.js実務テクニック

1. Vuex設計パターン

javascript

// モジュール構造の設計

```
modules: {  
  agency: { // 代理店情報モジュール  
    namespaced: true, // 名前空間有効化  
    state: {}, // 状態データ  
    actions: {}, // 非同期アクション  
    mutations: {}, // 状態変更  
    getters: {} // 状態取得  
  },  
  user: { // ユーザー情報モジュール (将来追加想定)  
    namespaced: true,  
    // ...  
  }  
}
```

2. テストコードの保守性

javascript

// 🌀 再利用可能なヘルパー関数

```
function getDateToSet(setAge) {  
  const currentYear = new Date().getFullYear() // 現在の年を取得  
  const setYear = String(currentYear - setAge) // 設定年を計算  
  const setDate = setYear + '-01-01' // 日付文字列作成  
  return { setYear, setDate } // オブジェクトで返却  
}
```

// 🌀 共通のテストデータ

```
const simulationParamSumame = () => {  
  const tmp = { /* テストデータ */ }  
  return JSON.parse(JSON.stringify(tmp)) // ディープコピーで返却  
}
```

3. 非同期処理のテスト

javascript

// async/await + flushPromises の組み合わせ

```
wrapper.find('.simulationBtn').trigger('click') // ボタンクリック  
await flushPromises() // 非同期処理完了を待機  
await wrapper.vm.$nextTick() // DOM更新完了を待機
```

開発効率の向上

- **Vuex導入**により状態管理が明確化
- **テスト環境改善**により不具合早期発見
- **命名統一**により可読性向上

保守性の向上

- **モジュール化**により影響範囲の局所化
- **テスト独立性**により安全なリファクタリング
- **一貫したコーディング規約**

品質の向上

- より本番環境に近いテスト
- 商品仕様変更への迅速対応
- 回帰テストの信頼性向上

🎓 まとめ：新人エンジニアへのアドバイス

学んだこと

1. **Vuex** - 大規模アプリでの状態管理の重要性
2. **テスト設計** - beforeEach/afterEachによる環境整備
3. **リファクタリング** - 段階的な改善アプローチ
4. **実務対応** - 商品仕様変更時のコード同期

次のステップ

1. **Vuex公式ドキュメント**を読んで理解を深める
2. **小さなプロジェクト**でVuexを実際に使ってみる
3. ****テスト駆動開発（TDD）****の考え方を学ぶ
4. **コードレビュー**で他の人のコードから学ぶ

実務で心がけること

- **コードは読まれるもの** - 半年後の自分が理解できるか？
 - **テストは保険** - 将来の変更に備えた投資
 - **段階的改善** - 一度に全てを変えずステップを踏む
 - **チーム開発** - 他の人も理解できるコードを書く
-

Vue.js先生より： 実務では今回のような「商品リニューアル」がよく発生します。その度に技術的な対応が必要になるのが現実です。今回学んだ**状態管理**、**テスト設計**、**段階的リファクタリング**は、どの現場でも役立つスキルです。ぜひ実践で活用してくださいね！💪

次回は「Vue 3 Composition API」を使った最新のコンポーネント設計について解説予定です。楽しみに！