Storybookをテスト仕様として使う

勉強会 第3回 📅 2026/3/13

今日のゴール

- Storybook が「**UIカタログ以上**」のものだと理解する
  - `play function` でインタラクションを記述する方法を知る
  - **Vitest addon** で Stories をテストとして実行する方法を知る

> *「Story を書くことが、テスト仕様を書くことになる」*

この勉強会のスコープ

この資料は以下を対象としています：

- フロントエンドのUIコンポーネント
- 状態・インタラクションの検証
- Storybook + Vitest によるテスト自動化

対象外：

- APIテスト / バックエンドテスト
- E2E（画面遷移・複数画面）
- 非同期処理の完全検証

まず問題：よくある Storybook の状態

Storybook は導入していても…

- コンポーネントを並べているだけ
- 見た目を眺めるだけ
- 壊れても仕様差分に気づきにくい

という状態になりやすい。

Storybook が本来できること

- UI状態を分けて表現できる
- 状態ごとの差を明示できる
- 操作手順も書ける
- テスト実行にもつなげられる

*Storybook は「**UI仕様の置き場所**」になれる*

今日の整理

Story = UI状態を表す仕様

play function = Storyに操作と期待を書く仕組み

Vitest addon = Storyをそのままテスト実行する仕組み

Story の3つの役割

役割	説明	今まで
UIカタログ	コンポーネントを一覧・確認する	✅ 使ってるはず
ドキュメント	Props の仕様を可視化する	✅ 使ってるはず
テスト仕様	状態・操作・期待値を定義する	🆕 今日のテーマ

Story = UI状態の定義

たとえばフォーム部品でも、これらは全部「別の仕様」：

- Empty
- Filled
- Invalid
- Disabled
- Submitting
- Success

→ **ユーザーが遭遇する意味のある状態** を Story にする

Story を作る観点

Story は **見た目ごと** ではなく
意味のある状態ごと に分けるのが大事です。

- 入力前 / 入力後
- 正常 / 異常
- 有効 / 無効
- 処理中 / 完了

→ **ユーザーにとって意味が変わるところで Story を分ける**

play function とは

Story に「**ユーザー操作のシナリオ**」を記述できる関数

```
export const SubmitSuccess: Story = {
  play: async ({ canvas, userEvent }) => {
    // 1. フォームに入力する
    await userEvent.type(canvas.getByLabelText('メールアドレス'), 'user@example.com')
    await userEvent.type(canvas.getByLabelText('パスワード'), 'password123')

    // 2. ボタンをクリックする
    await userEvent.click(canvas.getByRole('button', { name: '送信する' }))

    // 3. 成功メッセージが表示されることを確認する
    await expect(canvas.getByText('送信しました！')).toBeInTheDocument()
  },
}
```

play があると何が変わるか

Story が「静的な状態」だけでなく、

- この操作をしたら
- この表示になる
- この状態遷移になる

という **振る舞い付きの仕様** になる。

> Storybook UI の **再生ボタン**を押すと、ブラウザ上でそのまま操作が動く

play だけでも便利、でもそれだけでは足りない

`play function` があると、Storybook 上で操作と期待をその場で確認できます。

ただし、それだけだと

- 継続的に実行しにくい
- 変更時の回帰確認が人頼みになる
- CI に乗せにくい

という限界があります。

Vitest addon とは

`@storybook/addon-vitest` を使うと、StoryをそのままVitestでテストとして実行できる

	test-runner（旧）	Vitest addon（新・推奨）
ベース	Jest + Playwright	Vitest + Playwright
Storybook の起動	必要	**不要**

Vitest 実行のイメージ

```
pnpm vitest run --project=storybook
```

```
✓ src/components/Button/Button.stories.ts (3 tests)
  ✓ Primary
  ✓ Disabled
  ✓ SubmitSuccess  ← play function がテストとして実行される！
```

> Story が そのまま テストになる・CI に組み込める 🎉

何が嬉しいのか

Storybook をテスト仕様として扱える

- Story に状態を置く
- `play function` に操作と期待を書く
- Vitest addon で継続実行する

すると、

- UI仕様と確認ロジックが近づく
- Story とテストの二重管理を減らせる
- 状態追加時の追従がしやすい

1つの Story から再利用できるもの

Story 定義ひとつから：

- ドキュメント
- Interaction Test（play function）
- アクセシビリティ確認
- Vitest 実行（CI）

まで再利用できる。**資産が散らばらない。**

Storybookテストの限界

Storybookだけではカバーできない領域：

- ページ遷移（ルーティング）
- Server Component / SSRの完全検証
- API連携の実環境検証
- パフォーマンス・負荷

→ E2E（Playwright）などと組み合わせるのが前提

まとめ

- **Story** = UI状態を表す仕様
- **play function** = その仕様に対する操作と期待
- **Vitest addon** = その仕様を自動実行する仕組み

> 「Story を書く」ことと「テスト仕様を書く」ことを**同じアクション** にできる

参考リンク

- play function | Storybook docs
- Vitest addon | Storybook docs