Reactコンポーネントにおける テスト手法の選択肢

TechFeed Experts Night#2
〜 フロントエンドフレームワーク特集

@__sakito__
サイボウズ株式会社 Design Technologist

自己紹介

- Sakito (@__sakito__)
- Design Technologist
- サイボウズ株式会社
  - kintone Design Team
  - Design System 作ってます

今日話すこと・話さないこと

- **話すこと**
  - React 周辺ライブラリの最新情報とか交えつつ、コンポーネントテスト手法のあれこれ

- **話さないこと**
  - Unit Test, Integration Test, E2E Test の説明
  - テストの設計方法
  - 詳しい実装方法

React Testing Library

React Testing Library - 概要

- React のコンポーネントテストを書くライブラリ
- Jest や Vitest などのテストランナーを使って jsdom 上で動く
  - テストランナーを使用しなくてもいい
- jsdom を使い、Node.js 上でエミュレートするのでブラウザ上で再現する

  より実行動作が早い
- eslint-plugin-testing-library もあります

React Testing Library - a11y

- DOM 取得する方法を getByRole 優先にすることで、a11y を担保しつつテストを書くことができる
- 注意：すべての場合に当てはまるわけではないので、公式ドキュメントのPriorityの項を見るのがおすすめ
  - https://testing-library.com/docs/queries/about/#priority

React Testing Library - v13

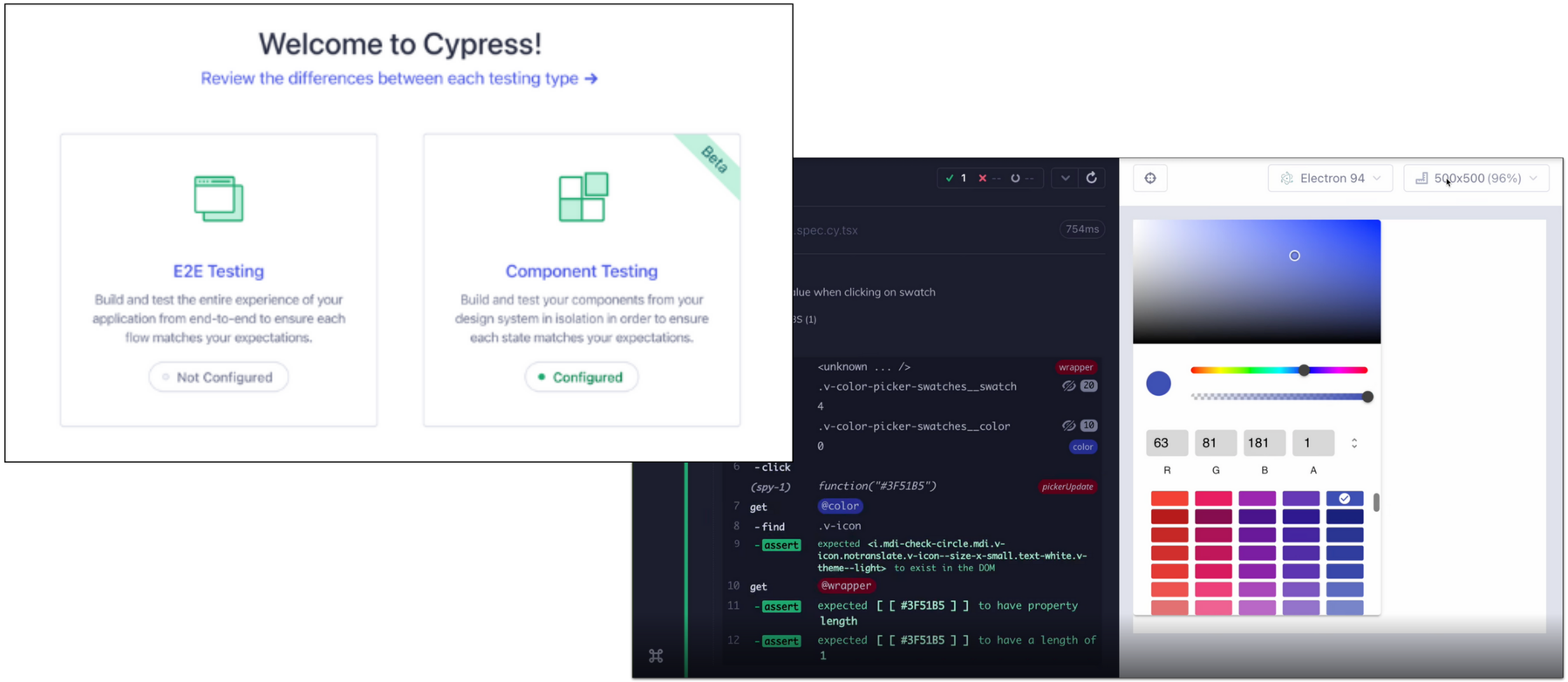
- React v18 に合わせたタイミングでリリースされた(2022年3月頃)
- React v17 以降のサポートが切れた
- React Hooks 単体をテストする renderHook 関数が追加された

Cypress, Playwright

Cypress, Playwright

- ブラウザ上で動かすテストツール
- コンポーネント単体でのテストができるようになってきている
  - Cypress は 2022年6月にリリースされた v10 で beta として使用できる
  - Playwright は 2022年5月にリリースされた v1.22.0 で Experimental として使用できる
- サーバーサイドと合わせたテストをする E2E 用の Test フレームワークではなく、単にブラウザ上で実行できるフレームワークとしての役割も

Cypress, Playwright



引用元：Announcing Cypress 10 with Component Testing Beta!
https://cypress.io/blog/2022/06/01/cypress-10-release/

jsdom (Node.js)とブラウザ上で実行する違い

- 実行環境が違う
  - jsdom (Node.js) : React Testing Library
  - ブラウザ : Cypress, Playwright
- jsdom は レイアウト諸々に関する API が使えない
  - 基本的なコンポーネントの機能テストは React Testing Library
  - レイアウトなどに関わる機能は Cypress や Playwright を使う
    - Mock がなくなることでテストの信頼度があり、コードの簡略になる

Storybookの活用

Storybook - 概要

- コンポーネント毎の描画などができる OSS
- コンポーネントカタログとしてのイメージが強いかもしれないが、

  Storybook 上で各コンポーネント(Story)のテストが行えるようになった
- Storybook チームが開発している Chromatic を使うことで、

  さらに便利になる

Storybook - @storybook/testing-react

- Storybook 上に定義した 1つ1つの Story を使用して、

  コンポーネントができる

- Storyで状態などは見えるようにしているので、

  それに対してテストすることができる

- @storybook/testing-library と一緒に使用する
  - Story + testing-library

Storybook - @storybook/testing-react

```tsx
// Button.stories.tsx

import { ComponentMeta, ComponentStoryObj } from '@storybook/react';
import { Button } from '..';

export default {
  title: 'Components/Button',
  component: Button,
  parameters: {
    docs: {
      page: mdx
    }
  }
} as ComponentMeta<typeof Button>;

// Storyを定義
export const NormalButton: ComponentStoryObj<typeof Button> = {
  args: {
    children: 'Click Me!',
    onClick: () => {
      console.log('クリック');
    },
    disabled: false
  }
};

export const Disabled: ComponentStoryObj<typeof Button> = {
  args: {
    ...NormalSizeLarge.args,
    disabled: true
  }
};
```

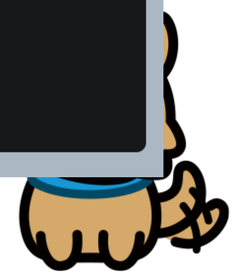
```tsx
// Button.test.tsx

import * as stories from '../stories/Button.stories';
import { composeStories } from '@storybook/testing-react';
import { waitFor, render, screen } from '@testing-library/react';

const { NormalButton, Disabled } = composeStories(stories);

it('マウスクリックでアクションが実行される', async () => {
  const onClickSpy = jest.fn();
  render(<NormalButton onClick={onClickSpy} />);
  const button = screen.getByRole('button');
  await waitFor(() => button);
  userEvent.click(button);
  expect(onClickSpy).toHaveBeenCalled();
});

it('マウスクリックでアクションが実行されない', async () => {
  const onClickSpy = jest.fn();
  render(<Disabled onClick={onClickSpy} />);
  const button = screen.getByRole('button');
  await waitFor(() => button);
  userEvent.click(button);
  expect(onClickSpy).not.toHaveBeenCalled();
});
```



Storybook - Interaction Testing

- Storybook 上で動作を再現できるテスト
  - @storybook/testing-library と @storybook/addon-interactions を使用する
  - play 関数を使う
- Focus, Active状態の確認や複雑なコンポーネントの動作を再現することで、コンポーネントの仕様が分かるようになる

Storybook - Interaction Testing

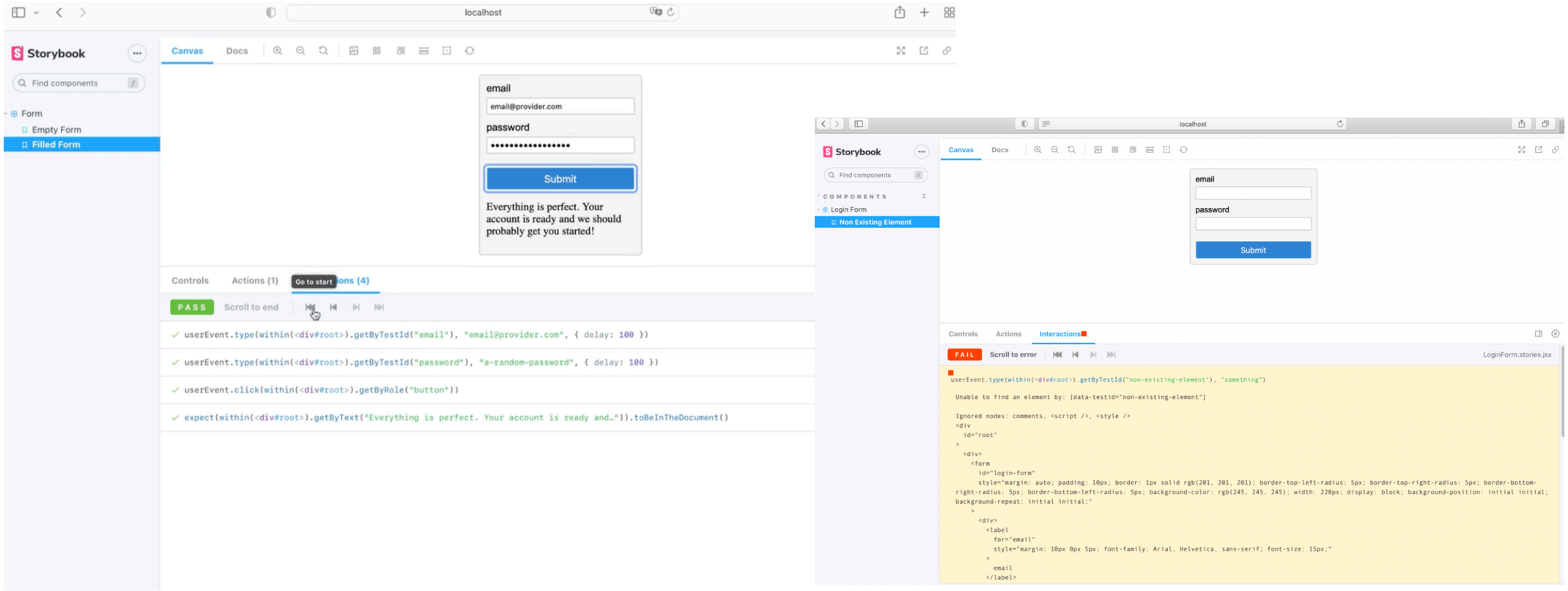
```tsx
import { ComponentMeta, ComponentStoryObj } from '@storybook/react';
import { within, userEvent } from '@storybook/testing-library';
import { Button } from '..';

export default {
  title: 'Components/Button/vrt',
  component: Button
} as ComponentMeta<typeof Button>;

// Active状態を再現
export const Active: ComponentStoryObj<typeof Button> = {
  play: async ({ canvasElement }) => {
    const canvas = within(canvasElement);
    userEvent.click(canvas.getByRole('button'));
  }
};

// Focus状態を再現
export const Focus: ComponentStoryObj<typeof Button> = {
  play: async ({ canvasElement }) => {
    const canvas = within(canvasElement);
    canvas.getByRole('button').focus();
  }
};
```

Storybook - Interaction Testing



引用元：Storybook - Interaction tests
https://storybook.js.org/docs/react/writing-tests/interaction-testing

Storybook - @storybook/test-runner

- Storybook上のコンポーネントテストをnpm sctiptで動かす
  - CI でも動かすことができる
  - Jest と Playwright で動く
- Interaction Test (Play 関数)があるものは、Interaction Test が通らなければ落ちる
- Interaction Test がないものは Story がレンダリングできるか検証されるので、Smoke Test になる

そのほか

- 時間がなくて割愛したもの達
- Storybook + Chromatic or reg-suit + Storycap を用いた Visual Regression Testing
- Axe を使用した a11y Test
  - jest-axe
  - Axe-Playwright

まとめ

- コンポーネントをテストする方法は様々ある
- テストの手札が多いほど、テスト設計の観点を増やすことができる
- コンポーネントやアプリケーションの性質、状況に合わせて使おう

おわり