mswでフロントエンドの
インテグレーションテストを実装する

Press Space for next page →

自己紹介

- 📝 飯野陽平（wheatandcat）
- 🏢 フリーランスエンジニア（シェアフル株式会社CTO）
- 💻 Blog: https://www.wheatandcat.me/
- 🛠️ 今までに作ったもの
  - memoir
  - ペペロミア
  - Atomic Design Check List

mswとは

- **msw**(Mock Service Worker)は、ネットワークレベルでAPIリクエストをインターセプトしてmockのデータを返すためのライブラリ
- **Service Worker** を使用しているので、別プロセスでローカルのサーバーを立ち上げる必要が無く手軽に利用できる
- フロントエンドの**テストコード**やStroybookなどでも利用可能
- デフォルトで**RESTful API**と**GraphQL**をサポートしている

- msw
- Service Worker とは
- msw Choose an API

インテグレーションテストとは？

- フロントエンドのテストは主に以下の4つに分類される
  - End to End
  - Integration
  - Unit
  - Static
- 上位レイヤーに行くほど以下の特徴がある
  - フィードバックの速度は遅い
  - 実装コストは高い
  - 信頼性は向上する

- The Testing Trophy and Testing Classifications

インテグレーションテストのこれまで ①

- msw導入前だと各APIを**jest**のmock機能を使用してAPIのコールメソッドを、すべてmockさせて実装させていく必要があった
- コード的には次のようになっていた

インテグレーションテストのこれまで ②

```
describe("components/pages/ItemDetail/Connected.tsx", () => {
  jest.spyOn(queries, "useItemQuery").mockImplementation((): any => ({
    loading: false,
    data: {
      item: {
        id: "1",
        title: "本を読む",
        categoryID: 2,
        date: "2021-01-01T00:00:00+09:00",
      },
    },
    error: undefined,
    refetch: jest.fn(),
  }));

  const wrapper = shallow(<Connected {...propsData()} />);

  it("正常にrenderすること", () => {
    expect(wrapper).toMatchSnapshot();
  });
});
```

インテグレーションテストのこれまで ②

```
describe("components/pages/ItemDetail/Connected.tsx", () => {
  jest.spyOn(queries, "useItemQuery").mockImplementation((): any => ({
    loading: false,
    data: {
      item: {
        id: "1",
        title: "本を読む",
        categoryID: 2,
        date: "2021-01-01T00:00:00+09:00",
      },
    },
    error: undefined,
    refetch: jest.fn(),
  }));

  const wrapper = shallow(<Connected {...propsData()} />);

  it("正常にrenderすること", () => {
    expect(wrapper).toMatchSnapshot();
  });
});
```

インテグレーションテストのこれまで ②

```
describe("components/pages/ItemDetail/Connected.tsx", () => {
  jest.spyOn(queries, "useItemQuery").mockImplementation((): any => ({
    loading: false,
    data: {
      item: {
        id: "1",
        title: "本を読む",
        categoryID: 2,
        date: "2021-01-01T00:00:00+09:00",
      },
    },
    error: undefined,
    refetch: jest.fn(),
  }));

  const wrapper = shallow(<Connected {...propsData()} />);

  it("正常にrenderすること", () => {
    expect(wrapper).toMatchSnapshot();
  });
});
```

モチベーション

- 今までmock作業で実装に時間の掛かったインテグレーションテストを簡単に実装できる状態にしたい
- また、実際の動作に伴ったシナリオテストを実装できる状態にしたい

実装に使用したライブラリ

- msw
- typed-document-node
- graphql-codegen-typescript-mock-data
- testing-library/react-native

実装PR

- typed-document-nodeへの移行
- mswを導入してGraphQLをモックしたテストコードを書く①
- mswを導入してGraphQLをモックしたテストコードを書く②

実装してみた ①

- まずは、typescript-react-apollo→typed-document-node移行を実装
- PR: typed-document-nodeへの移行
- 実装の紹介
  - こちら、次のPRで使用するgraphql-codegen-typescript-mock-dataを使用しやすくするために移行
  - **typed-document-node**は特定のGraphQLクライアントに依存せずにGraphQLのtypeを自動生成してくれるcodegenのプラグイン

実装してみた ②

- 次に、mswとgraphql-codegen-typescript-mock-dataを導入
- PR: mswを導入してGraphQLをモックしたテストコードを書く①
- 実装の紹介
  - **graphql-codegen-typescript-mock-data**は、GraphQLのSchema情報からダミーデータを生成してくれるライブラリ
  - これを利用することでmockデータを保守を自動生成で補うようにする
    - 参考コード
  - mswは以下の部分でテスト時に起動させる
    - 参考コード
  - テストは以下の通りに記載
    - 参考コード

実装してみた ③

- 最後に、前のPRからの追加で、より正確なシナリオテストを実装
- PR: mswを導入してGraphQLをモックしたテストコードを書く②
- 実装の紹介
  - アイテムの更新のシナリオテストを追加
    - 参考コード
  - testing-library/react-nativeを使用して各入力を設定
    - 参考コード
  - 入力に設定した値と更新のAPI時に設定された値とGraphQLのvariablesを比較して一致しているかテスト
    - 参考コード

まとめ

- mswを使えば気軽にインテグレーションテストを書けて、かなり良い。
- graphql-codegen周りの自動生成周りのツールと相性が良い
- テストから仕様が把握できるコードにしやすくなった

ご清聴ありがとうございました