React Native Test case 작성하기

목차

React에서의 Test Case

- React에서는 크게 Unit, Component, E2E 테스트로 나뉘어진다.
- Unit 테스트는 각 함수(메소드)에 대해서 함수가 동작 했을 때의 값을 예상(expect)하여 예상한 값과 실제 발생한 값이 같은지 비교한다.
- Component 테스트는 Component 단위로 테스트한다.
    - 1) mock props를 넘겨 테스트하고자하는 Component에서 원하는대로 동작하는가
    - 2) click, press와 같은 event를 발생시켜 의도한 값이 변경 혹은 반영 되었는가
    - 3) 해당 Component의 SnapShot을 찍은 후, Snapshot과 현재 Component에 변화가 있는가, 있다면 내가 의도한 것인가
    - 등등
- E2E 테스트는 End-to-End 테스트로, 사용자가 직접 UX/UI를 이용해보면서 network fetch 등을 확인하는 테스트이다. QA 할 때 유용할 것 같다.

React Native에서의 Test Case

사용하는 테스트 라이브러리

- Unit 테스트 → Jest
- Component 테스트 → Enzyme
- E2E 테스트 → Cypress

- React Native에서 가장 크게 다르다고 느껴진 테스트는 Component 테스트였다. 그 이유는 다음과 같다.
    - React Native에서는 React와 다른 HTML Tag를 사용한다.
    - React Native에서는 React와 다른 event명을 사용한다.
- 문제
    - React에서는 tag name, id, class명 등을 통해 Component에서 테스트에 필요한 아이템을 찾을 수 있고, 대부분은 tag name을 find()하여 사용하였다. React Native에서는 tag name을 통해 찾았을 때, 잘 찾아지지 않았다

```
•     const wrapper = mount(<SignInScreen />);
      wrapper.find('TouchableOpacity').first().props().onPress();
  //tag name
```

    - React에서는 click event를 enzyme에서 simlulate("click")을 통해 발생시켰는데, React Native에서 "click" 대신 "press"를 넣어 event를 발생시켜보았지만 찾을 수 없는 event명이라고 하였다.

```
•     const wrapper = mount(<SignInScreen />);
      const plusButton = wrapper.find('TouchableOpacity').
  first();
      plusButton.simulate('click'); //click
```

- 해결
    - React Native만을 위한 해결방법은 아니지만 내가 원하는 tag에 testID props를 추가하여 find()할 때, find({testID:'[testID]'})로 찾았다. 이는 같은 tag name을 사용하는 테스트하고 싶지 않은 tag를 배제할 수 있어서 유용하다고 생각한다.

```
•    const wrapper = mount(<SignInScreen />);
      wrapper.find({ testID: 'plusTest' }).first().props().
  onPress(); //testID
```

- React Native를 사용하는 우리 회사의 서비스는 Button이 아닌 TouchableOpacity를 사용하고, 이는 onPress를 통해 동작한다. 따라서 simulate가 아닌 해당 tag의 props로 접근하여 onPress()를 직접 호출하였다.

```
• const wrapper = mount(<SignInScreen />);
  wrapper.find({ testID: 'plusTest' }).first().props().onPress();
  //
```

Test Case Setting 및 실행

1. Jest → Unit 테스트 진행

<Setting>

- Jest는 ReactJs, React Native 내장 테스트 라이브러리이다.
- Typescript, Babel을 위한 Setting을 진행하였다.
- `yarn add -D jest @types/jest`
- `yarn add -D babel-jest babel-plugin-module-resolver @babel/core @babel/preset-env @babel/preset-typescript`

```
• //babel.config.js
  module.exports = {
    presets: [
        'module:metro-react-native-babel-preset',
      [
          '@babel/preset-env', {targets: {node: 'current'}}
      ],
      '@babel/preset-typescript'
    ],
    plugins: [
      [
        'module-resolver',
        {
          root: ['./src'],
          extensions: [
            '.ios.ts',
            '.android.ts',
            '.ts',
            '.ios.tsx',
            '.android.tsx',
            '.tsx',
            '.jsx',
            '.js',
            '.json',
          ],
          alias: {
            ...
          },
        },
      ],
    ],
  };
```

```
//jest.config.js
module.exports = {
    preset: 'react-native',
    moduleFileExtensions: ['ts', 'tsx', 'js', 'jsx', 'json',
'node'],
    setupFilesAfterEnv: ['./node_modules/jest-enzyme/lib/index.
js'], //enzyme
    testEnvironment: 'jsdom',
    transform: {
        '^.+\\.(js|jsx)?$': 'babel-jest',
        '^.+\\.(ts|tsx)?$': 'babel-jest',
    },
    verbose: true,
    moduleNameMapper: {
        '^@/(.*)$': '<rootDir>/$1',
        ...
    },
    cacheDirectory: ".jest/cache",
    snapshotSerializer : "enzyme-to-json/serializer" //enzyme
};
```

<실행>

./[test할 파일 경로]/__test__/[test할 파일명].test.ts

예시) Camel Case util test

```
import CamelCase from '@utils/CamelCase';

const SnakeCaseData = {
  user_id: 4,
  common_profile: {
    name: 'jiwoo',
    phone_number: '010-8950-6343',
  },
};
const CamelCaseData = {
  userId: 4,
  commonProfile: {
    name: 'jiwoo',
    phoneNumber: '010-8950-6343',
  },
};
const WrongData = {
  userId: 4,
  commonProfile: {
    name: 'jiwoo',
    phone_number: '010-8950-6343',
  },
};
describe('CamelCase Test', () => {
  it('changed?', () => {
    expect(CamelCase(SnakeCaseData)).toEqual(CamelCaseData); //passed
  });
  it('wrong?', () => {
    expect(CamelCase(SnakeCaseData)).toEqual(WrongData); //fail
  });
});

export {};
```

2. Enzyme → Component 테스트 진행

<Setting>

- React Native 0.66.9는 React 17 버전을 사용한다. 하지만 아직 정식으로 enzyme가 지원하는 버전이 없어서 다음과 같이 설치했다.
- `yarn add -D enzyme enzyme-to-json jest-enzyme @wojtekmaj/enzyme-adapter-react-17 @types/enzyme`
- enzyme-to-json은 snapshot을 찍을 때 jest와 serialize하기 위해 설치하였다.

<실행>

예시) Typo component SnapShot, Props 테스트

```
import 'react-native';
import React from 'react';
import Typo from '@components/ui/Text/Typo';
import { configure, mount, shallow } from 'enzyme';
import Adapter from '@wojtekmaj/enzyme-adapter-react-17';

configure({ adapter: new Adapter() });

describe('Typo test ', () => {
  //snapshot test
  it('matches Snapshot', () => {
    const wrapper = mount(<Typo fontType={'Heading_04'}></Typo>);
    expect(wrapper).toMatchSnapshot();
  });
  //component test
  it('Render?', () => {
    const wrapper = shallow(<Typo fontType={'Heading_03'}></Typo>);
    expect(wrapper.length).toEqual(1);
  });
  it('fontType check?', () => {
    const wrapper = mount(<Typo fontType={'Heading_04'}></Typo>);
    expect(wrapper.props().fontType).toBe('Heading_04'); //font type
check
    const Text = wrapper.find('Text');
    expect(Text.first().text()).toEqual('');  //text check
  });
});

export {};
```

Hooks로 구현한 component에서 테스트하기

```
import 'react-native';
import React from 'react';
import { configure, mount, shallow } from 'enzyme';
import Adapter from '@wojtekmaj/enzyme-adapter-react-17';
import SignInController from '@screens/SignInScreen';
import '@react-navigation/native';
import { SignInScreen } from '~/screens';

jest.mock('@react-navigation/native');

configure({ adapter: new Adapter() });

// With this implementation,
// we are mocking React.useState to return an Array
// with the initial value passed to the method and a jest mock function.
// This will set the states setter to our mock function
// and allow us to test that it was called with the expected value to
set state to.

describe('SignInController test ', () => {
   const setState = jest.fn();
   const useStateSpy = jest.spyOn(React, 'useState');
   useStateSpy.mockImplementation((init) => [init, setState]);

   afterEach(() => {
     jest.clearAllMocks();
   });


   it('useState press', () => {
      jest.spyOn(React, 'useState').mockImplementation(useStateMock);
      const wrapper = shallow(<SignInScreen />).dive();
      wrapper.find({ testID: 'plusTest' }).props().onPress();

      expect(setState).toHaveBeenCalledTimes(1);
    });

});

export {};
```

onPress를 하면 state가 +1되는 것이다. 이 때 setState를 mockFunction으로 가져오고, React.useState를 spy하여 useState를 따라하는 useStateSpy를 만들어서 테스트한다. 그러면 useState처럼 테스트 할 수 있다.