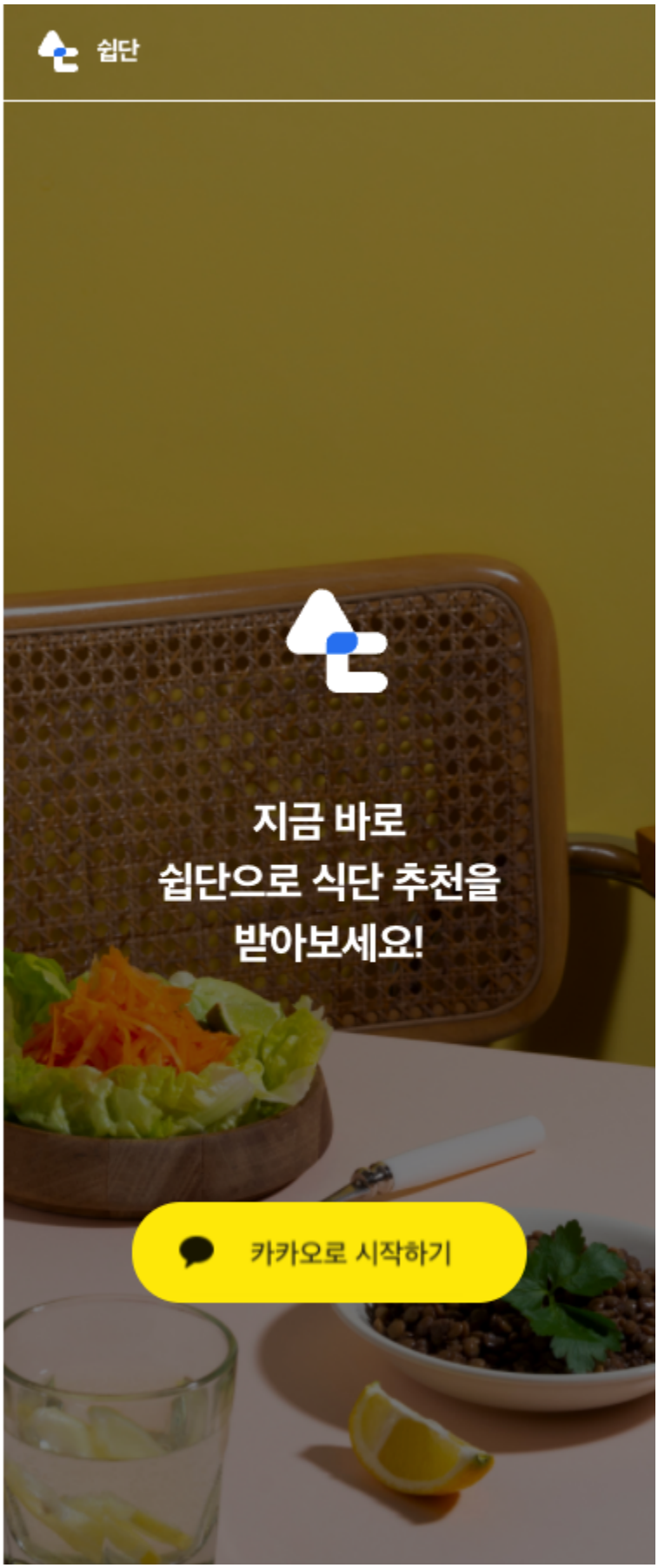
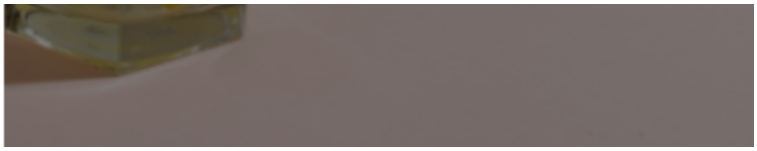
쉽단 SignIn Page TDD concept





1. Container와 Presenter를 분리하자(SoC)
2. mock 함수를 __mocks__ 에 모으자

어떤 것을 테스트할 것인가?

- SignIn Container(Form) → 렌더링이 잘 되는가?
- ***SignIn Presenter → BDD(Behavior Driven Development) → img url이 있을 경우와 없을 경우에 대한 rendering test***
- api → background image API test
- ***SignInButton Presenter → FireEvent로 함수가 호출 되는가?***

SignIn Presenter

1. img Url이 있을 경우 → 위의 이미지와 같은 화면이 렌더링 되어야한다. → 쉽단, 지금 바로 등등의 text가 있다 && img tag의 src attribute의 url이 props로 전달한 img의 url과 같을 것이다.
2. img Url이 없을 경우(string length가 0일 경우) → 로딩 화면일 렌더링 되어야한다. → Loading Wrapper가 있을 것이다.

<Concept>

- 1번에서의 img tag와 2번에서의 Loading Wrapper를 testId로 얻어오자 → HTML tag에 data-testid="[TEST_ID]" 지정

[signIn/Presenter.tsx]

```
...
interface Props {
  img: string;
  handleToken: (t: string) => void;
}

const Presenter: React.FC<Props> = props => {
  const { img, handleToken } = props;

  //img url
  return img ? (
    <>
      <Wrapper>
        <Header>
          <ShipdanLogoColor width={27} height={27} />
          <HeaderText>
            <Typo fontType="Heading_04" color={Color.white_100}>

            </Typo>
          </HeaderText>
        </Header>
        <Body>
          <IconWrapper>
            <ShipdanLogoColor width={67.5} height={67.5} />
          </IconWrapper>
          <Title>
            <Typo fontType="Heading_01" color={Color.white_100}>
```

```
              </Typo>
              <Typo fontType="Heading_01" color={Color.white_100}>

              </Typo>
              <Typo fontType="Heading_01" color={Color.white_100}>
                !
              </Typo>
            </Title>

            <KakaoSignInSmall handleToken={handleToken} />
          </Body>
        </Wrapper>
        <BackgroundImg data-testid="background-img" src={img} />
      </>
    ) : (
    //img url
      <LoadingWrapper data-testid="loading-wrapper">
        <Spinner />
      </LoadingWrapper>
    );
};
....
export default Presenter;
```

[Presenter.test.tsx]

```
import React from 'react';
import { cleanup, render } from '@testing-library/react';
import 'jest-styled-components';
import Presenter from '@components/signin/Presenter';
import { expect } from '@jest/globals';

jest.mock('react');
jest.mock('react', () => ({
  ...jest.requireActual('react'),
  useContext: jest.fn(), //useContext mocking
}));
jest.mock('@components/signin/SignInBtn/KakaoSignInSmall', () => {
  return () => <div />;
}); //Component mocking

describe('signIn Presenter', () => {
  const img = 'https://www.gstatic.com/webp/gallery3/1.sm.png'; // img
url

  const handleTokenMock = jest.fn(); //handleToken props mock function

  const renderPresenter = (imgUrl: string) => {
    return render(<Presenter img={imgUrl} handleToken={handleTokenMock}
/>);
  };

  afterEach(cleanup);

  //img url
  it('rendering with img url', () => {
    const presenter = renderPresenter(img);
    const bg = presenter.getByTestId('background-img');

    expect(bg.getAttribute('src')).toBe(img);
    expect(presenter.getByText(''));
    expect(presenter.getByText(' '));
  });

  //img url
  it('rendering without img url', () => {
    const presenter = renderPresenter('');
    const loading = presenter.getByTestId('loading-wrapper');
    expect(loading).toBeTruthy();
  });
});
```

- 난관 : Presenter의 KakaoSignInSmall에서 window.Kakao가 undefined이어서 error 발생

→ KakaoSignInSmall 컴포넌트를 로 mocking

SignInButton Presenter

- Button을 click하면 props로 전달 받은 onClick 함수가 실행될 것이다.

[SignInBtn/Presenter.tsx]

```
...
interface Props {
  mode: 'kakao' | 'email' | 'apple' | 'kakaoSmall';
  onClick?: () => void;
}

const Presenter = (props: Props) => {
  const { mode, onClick } = props;

  const getIcon = () => {
   ...
  };

  return (
    <Container data-testid={'signin-button'} onClick={onClick}>
      {getIcon()}
    </Container>
  );
};
...
export default memo(Presenter);
```

[SignInBtn.test.tsx]

```
import React from 'react';
import { cleanup, fireEvent, render } from '@testing-library/react';
import 'jest-styled-components';
import Presenter from '@components/signin/SignInBtn/Presenter';

jest.mock('react');
jest.mock('react', () => ({
  ...jest.requireActual('react'),
  useContext: jest.fn(),
  useEffect: jest.fn(), //useEffect undefined window.Kakao  mocking
}));
describe('KakaoSignIn Presenter', () => {
  const onClickMock = jest.fn(); //     onClick function mocking

  const renderPresenter = () => {
    return render(<Presenter onClick={onClickMock} mode={'kakaoSmall'}
/>);
  };

  afterEach(cleanup);

  it('fire click', () => {
    const signIn = renderPresenter();
    const signInButton = signIn.getAllByTestId('signin-button');
    fireEvent.click(signInButton[0]);
    fireEvent.click(signInButton[0]);

    expect(onClickMock).toBeCalledTimes(2);
  });
});
```

- 난관 : KakaoSignInSmall에서 useEffect()에서 사용하는 window.Kakao가 undefined이어서 error 발생 → useEffect를 mocking

<Concept>

render()한 Presenter에서 signin-button testId를 가지는 div를 onClick을 fireEvent하면, fire한 횟수 만큼 onClickMock()이 실행될 것이다.

getLandingPageImg API

- axios를 통해 만든 axiosInstance를 mocking하여 checking
- axios-mock-adapter library를 이용하여 axios mocking

[default.test.ts]

```
jest.unmock('axios');
import React from 'react';
import MockAdapter from 'axios-mock-adapter';
import axios from 'axios';
import '@testing-library/jest-dom';
import API from '@api/api';

jest.mock('react');

const mockAxios = new MockAdapter(API, {
  delayResponse: 200,
  onNoMatch: 'throwException',
});
const data = {
  img: 'https://www.gstatic.com/webp/gallery3/1.sm.png',
  result: {
    img: 'https://www.gstatic.com/webp/gallery3/1.sm.png',
  },
};
// data
describe('axios', () => {
  beforeAll(() => {
    mockAxios.onGet('spark/landing').reply(200, data);
    //'spark/landing' api GET method mocking
  });

  afterAll(() => {
    mockAxios.restore();
  });

  it('getLandingPageImg GET success', async () => {
    await expect(API.get('spark/landing')).resolves.toMatchObject({
      status: 200,
      data,
    });
  });
});
```