2023.04.16 프론트 엔드의 테스트 코드

테스트 코드를 짜는 이유

1. 리팩토링 전후의 결과가 일치함을 보장해 줍니다.
2. 새로운 기능이 추가되거나 기존의 기능이 변경되었을 때, 다른 기능들이 제대로 동작하는지 확인할 수 있어 유지보수성을 향상시켜 줍니다.

테스트 방법론

1. TDD(Test-Driven Development)는 **테스트 주도 개발**을 의미하며, 요구 사항을 검증하는 테스트 케이스를 먼저 작성한 후, 이를 통과할 수 있는 코드를 작성하고 리팩토링하는, 선 테스트 후 개발 순의 개발 방법론입니다.
2. BDD(Behavior-Driven Development)는 **행동 주도 개발**을 의미하며, TDD를 근간으로 하여 파생되었습니다. TDD와 유사하지만 비즈니스 요구사항에 더 가까운 용어를 사용하여 개발을 진행합니다.

테스트의 유형

1. 정적(Static) 테스트
    a. **코드를 실행시키지 않고 테스트**하는 것을 말합니다.
    b. Type Error, Reference Error 등, 개발자의 실수를 미연에 방지할 수 있습니다.
    c. 프론트엔드에서는 TypeScript로 타입을 검사하거나, ESLint로 사용하지 않는 변수를 찾는 등의 행위가 정적 테스트에 포함됩니다.
2. 단위(Unit) 테스트
    a. 각 **모듈을 단독 실행 환경에서 독립적으로 테스트**하는 것을 말합니다.

b. 의존성이 있는 코드와 함께 테스트하는 **Sociable 테스트**, 테스트 더블(더미, 페이크, 목 등)로 의존성이 있는 코드를 대체하여 테스트하는 **Solitary 테스트**가 있습니다.

c. 프론트엔드에서는 특정 컴포넌트를 렌더링하여 확인하는 것을 예로 들 수 있습니다.

d. 이 때 Sociable 테스트는 자식 컴포넌트를 포함하여 렌더링하는 것, Solitary 테스트는 자식 컴포넌트를 모킹하여 렌더링하는 것이라 할 수 있습니다.

> 모킹이란, 테스트를 수행할 모듈과 연결된 외부의 다른 서비스나 모듈을, 실제 사용되는 모듈이 아닌 이를 흉내낸 가짜 모듈을 생성하는 것을 의미합니다.

3. 통합(Integration) 테스트

a. **둘 이상의 모듈이 실제로 연결된 상태를 테스트**하는 것을 말하며, 모듈 간의 연결에서 발생하는 에러를 검증할 수 있습니다.

b. 의존성이 있는 코드와 함께 테스트하는 **Broad 테스트**, 테스트 더블로 의존성이 있는 코드를 대체하여 테스트하는 **Narrow 테스트**가 있습니다.

c. 프론트엔드에서는 API와 UI의 상호 작용이나 State와 UI의 상호 작용 등을 테스트하는 것을 예로 들 수 있습니다.

d. 실제 API를 호출하면 Broad 테스트, Response를 모킹하거나 가상 API 서버를 이용하면 Narrow 테스트가 되겠습니다.

4. E2E 테스트(End to End)

a. **실제 사용자의 입장 및 환경에서 테스트**하는 것을 말하며, 프론트엔드에서는 이는 즉 브라우저에서 테스트하는 것을 의미합니다.

b. 실제 상황에서 발생할 수 있는 에러를 검출 가능하며, Web API 사용이 가능합니다. 또한 테스트 코드가 내부 구조에 영향을 받지 않으므로 코드를 변경해도 비교적 잘 깨지지 않습니다.

c. 단위 테스트나 통합 테스트에 비해 실행 속도가 느리며 실험 환경을 전부 통제할 수 없으므로 테스트 결과를 신뢰하기 어려울 수 있습니다.

테스트의 대상

1. 시각적 요소

a. **HTML, CSS의 구조나 계산 결과**가 의도대로 나타나는지를 테스트합니다.

b. **스냅샷 테스트**는 어떤 기능의 예상 결과를 미리 포착해 두고 실제 결과와 비교하는 테스트 기법입니다. 스냅샷 테스트를 제공해주는 도구로는 **Jest**가 있습니다.

   i. Jest는 `toMatchSnapshot` 또는 `toMatchInlineSnapshot`을 사용하여 데이터의 스냅샷을 저장할 수 있습니다. 이를 통해 렌더링 된 컴포넌트 출력을 저장하고 컴포넌트 출력이 변경되면 이를 감지해 줍니다.

c. **시각적 회귀 테스트(Visual Regression Test)**는 컴포넌트가 실제로 브라우저에서 의도대로 렌더링 되는지 테스트합니다. 실제 브라우저에서 스냅샷을 캡쳐하고 현재 테스트 중인 화면을 캡쳐하여 두 이미지를 비교합니다. 시각적 회귀 테스트를 제공하는 도구로는 **storybook**이 있습니다.

   i. 스냅샷 테스트와 유사해 보이지만, 코드 형태로 저장되어 **JSON 데이터, 텍스트, 컴포넌트 렌더링 결과 등의 오류를 체크하는 스냅샷 테스트**와 달리, **스타일, 레이아웃, 디자인 등의 시각적 요소의 오류를 체크**합니다.

2. 이벤트 처리

   a. click, type, keyboard, upload, hover, tab 등 유저가 실제로 웹페이지를 사용하며 발생시키는 이벤트를 메서드로 제공하여, 실제 **사용자가 컴포넌트를 사용하는 것처럼 테스트**를할 수 있습니다. **Testing Library**가 대표적인 도구입니다.

      i. Testing Library는 특정 state나 method를 테스트하는 것이 아닌, 컴포넌트의 렌더링 결과에 접근하여 테스트를 수행합니다. 따라서 내부 로직이 변경되어도 테스트가 깨지지 않습니다.

3. API 통신

   a. **API 서버와의 통신을 테스트**합니다.

   b. 실제 API 서버를 이용하는 방법이 있으며, 이는E2E 테스트에서 주로 사용됩니다.

   c. 테스트용 API 서버를 구축하거나 API 클라이언트를 모킹하는 방법이 있습니다. 간단하게 모듈을 모킹할 수 있는 Jest가 대표적인 도구로 사용됩니다.

테스트 환경

1. 브라우저 환경

   a. 장점

      i. 모든 Web API에 접근이 가능합니다.

        ii. 서로 다른 브라우저에서 테스트할 수 있으므로 브라우저 호환성 및 기기 호환성 테스트 진행이 가능합니다.
    b. 단점
        i. 브라우저가 Node.js보다 무겁기 때문에 실행 속도가 느립니다.
        ii. 브라우저 런처를 별도로 설치해야 합니다.
2. Node.js 환경
    a. 장점
        i. 브라우저 환경보다 속도가 빠릅니다.
    b. 단점
        i. Web API나 DOM 접근 API를 사용할 수 없습니다.
            1. Jest의 jsdom과 같은 가상 DOM을 제공하는 테스트 도구가 있으나, 페이지 네비게이션이나 레이아웃 등에 대한 테스트는 여전히 진행할 수 없습니다.