통합 테스트 대상 선정하기

실무에 바로 적용하는 프런트엔드 테스트

이재성

사용 브랜치

shopping-mall-integration-test

$ git checkout shopping-mall-integration-test

단위 테스트는 무엇을 검증할까?

✔ 의존성이 없거나 일부의 외부 의존성만 존재하는 모듈

✔ 작은 단일 컴포넌트, 유틸 함수, 리액트 훅

통합 테스트는 무엇을 검증할까?

✔ API·상태 관리 스토어·리액트 컨텍스트 등 다양한 요소들이 결합된 컴포넌트가 특정 비즈니스 로직을 올바르게 수행하는지 검증

✔ 주로, 컴포넌트간의 상호작용·API 호출 및 상태 변경에 따른 UI 변경 사항 검증

메인 페이지의 비즈니스 로직

- 네비게이션 영역의 로그인 여부에 따른 동작
- API를 통해 필터의 카테고리 데이터를 올바르게 렌더링 하는지
- 필터 항목을 수정했을 때 올바르게 반영되는지 검증
- API 응답에 따라 상품 리스트가 적절하게 렌더링 되는지
- …

메인 페이지의 비즈니스 로직

- 네비게이션 영역의 로그인 여부에 따른 동작
- API를 통해 필터의 카테고리 데이터를 올바르게 렌더링 하는지
- 필터 항목을 수정했을 때 올바르게 반영되는지 검증
- API 응답에 따라 상품 리스트가 적절하게 렌더링 되는지
- …

다양한 기능 검증을 위한 API, 상태 관리 스토어 모킹 필요

모킹(Mocking)은,

모킹은 테스트를 **독립적으로 분리**하여
효과적으로 검증할 수 있게 도와주지만,

모킹(Mocking)은,

모킹은 테스트를 **독립적으로 분리**하여
효과적으로 검증할 수 있게 도와주지만,

지나치게 많은 모킹은
테스트 신뢰성을 저하시키며 변경에 취약하다.

거대한 통합 테스트는,

모킹 코드 증가

일부 컴포넌트만 수정해도 많은 테스트가 깨질 수 있음

유지 보수에 많은 비용이 듦

지나친 모킹으로 **테스트 신뢰성 저하**

비즈니스 로직은 뭘까?

✔ 프로그램의 핵심 기능을 구현하는 코드

✔ 즉, 사용자가 원하는 결과를 얻기 위한
계산·처리·의사 결정을 수행하는 코드

비즈니스 로직을 통해

서비스의 정책, 절차, 규칙 등을 코드로 구현

비즈니스 로직을 기준으로 통합 테스트를 작성하면,

✔ 서비스의 핵심 비즈니스 로직을 **독립적인 기능 관점**에서 효율적으로 검증

✔ 테스트를 **명세 자체로 볼 수 있어** 앱을 이해하는데 큰 도움을 받을 수 있음

✔ 불필요한 단위 테스트를 줄여 **유지 보수 측면**에서도 좋음

메인 페이지의 비즈니스 로직을 크게 보면

1. 네비게이션 바 영역: 사용자 로그인 여부에 따른 UI 렌더링 및 상호작용(로그인, 로그아웃)
2. 상품 검색 영역: 필터 요소에 따른 검색 조건 설정
3. 상품 리스트 영역: 검색 결과에 따른 상품 리스트 렌더링 및 버튼 클릭에 따른 상호작용

메인 페이지의 비즈니스 로직을 크게 보면

1. 네비게이션 바 영역: 사용자 로그인 여부에 따른 UI 렌더링 및 상호작용(로그인, 로그아웃)
2. 상품 검색 영역: 검색 필터 필드 추가 및 변경
3. 상품 리스트 영역: 상품 리스트 렌더링 형태 변경

모두 하나의 비즈니스 로직 안에서 처리한다면..? 🤔

메인 페이지의 비즈니스 로직을 크게 보면

1. 네비게이션 바 영역 - 사용자 로그인
2. 상품 검색 영역 - 상품 검색 조건 변경
3. 상품 리스트 영역 - 상품 리스트 렌더링

✔ 각 비즈니스 로직에서 중요한 기능 검증

✔ 각 영역 별로 필요한 부분만 모킹

비즈니스 로직을 고려한 통합 테스트 범위

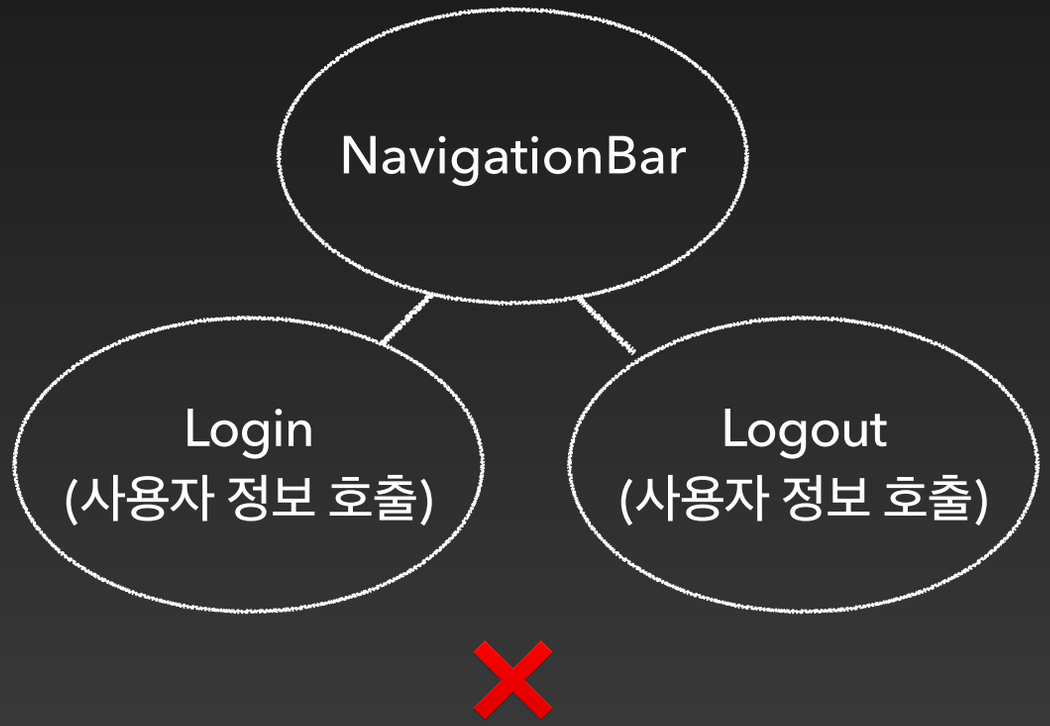
- ✔ 가능한 모킹을 하지 않고 최대한 앱의 실제 기능과 유사하게 검증하기

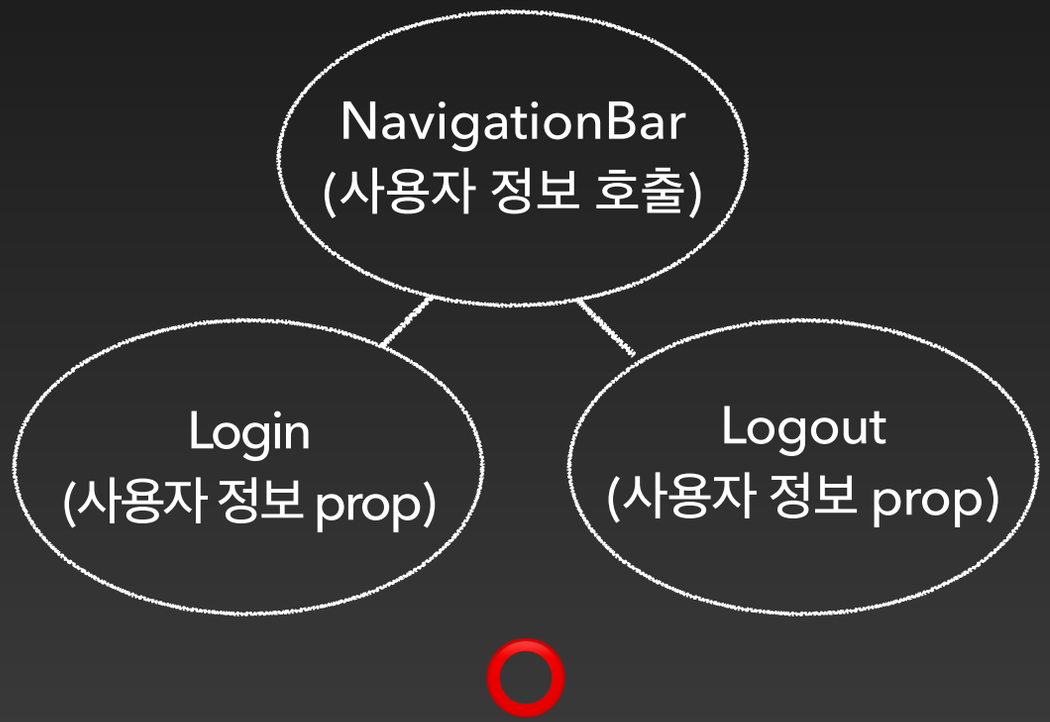
비즈니스 로직을 고려한 통합 테스트 범위

- ✔ 가능한 모킹을 하지 않고 최대한 앱의 실제 기능과 유사하게 검증하기
- ✔ 비즈니스 로직을 처리하는 상태 관리나 API 로직은 상위 컴포넌트로 응집해 관리

상위 컴포넌트로 응집해 관리

ex> 로그인, 로그아웃 버튼에서 각각 사용자의 상태를 조회 X
상위 컴포넌트에서 상태를 관리해 로직 관리 O





비즈니스 로직을 고려한 통합 테스트 범위

- ✔ 가능한 모킹을 하지 않고 최대한 앱의 실제 기능과 유사하게 검증하기
- ✔ 비즈니스 로직을 처리하는 상태 관리나 API 로직은 상위 컴포넌트로 응집해 관리
- ✔ 변경 가능성을 고려해 여러 도메인 기능이 조합된 비즈니스 로직은 나누어 통합 테스트를 작성

비즈니스 로직을 고려한 통합 테스트 범위

- ✔ 가능한 모킹을 하지 않고 최대한 앱의 실제 기능과 유사하게 검증하기
- ✔ 비즈니스 로직을 처리하는 상태 관리나 API 로직은 상위 컴포넌트로 응집해 관리
- ✔ 변경 가능성을 고려해 여러 도메인 기능이 조합된 비즈니스 로직은 나누어 통합 테스트를 작성
  ex) 메인 홈 페이지 → 사용자 로그인 + 상품 검색 + 상품 리스트

장바구니 페이지의 비즈니스 로직을 크게 보면

1. 상품 리스트 영역: 상품 리스트 렌더링 및 수량 수정,
   삭제 버튼 클릭에 따른 상호 작용
2. 가격 계산 영역: 모든 상품의 수량과 가격을 계산

정리

- 통합 테스트에서는 구성된 비즈니스 로직을 적절한 단위로 나눠 컴포넌트 집합을 검증해야 한다
- 비즈니스 로직을 기준으로 통합 테스트를 나눌 때는
  - 가능한 한 모킹을 하지 않고 실제와 유사하게 검증한다
  - 비즈니스 로직을 처리하는 상태 관리나 API 호출은 상위 컴포넌트로 응집한다
  - 변경 가능성을 고려해 여러 도메인의 기능이 조합된 비즈니스 로직은 나눠 검증한다
- 메인 페이지는 네비게이션 바·상품 검색·상품 리스트 영역으로 나눠 테스트를 작성한다
- 장바구니 페이지는 상품 리스트·가격 계산 영역으로 나눠 테스트를 작성한다
- 이를 통해 컴포넌트간 결합도를 낮추고 견고한 설계를 통해 유지 보수하기 좋은 코드를 작성할 수 있다