7/20 ~ 7/24 3주차 - 1

테스트 코드에 대해서 (3) - Mock 과 Unit 테스트

Mock

Mock 객체란 모듈의 겉모양이 실제 모듈과 비슷하게 보이도록 만든 가짜 객체.

모듈이 필요로 하는 의존성이 테스트 케이스 작성을 어렵게 만드는데, 이 의존성을 단절하기 위해 Mock 객체가 사용됨. (이해안됨..)

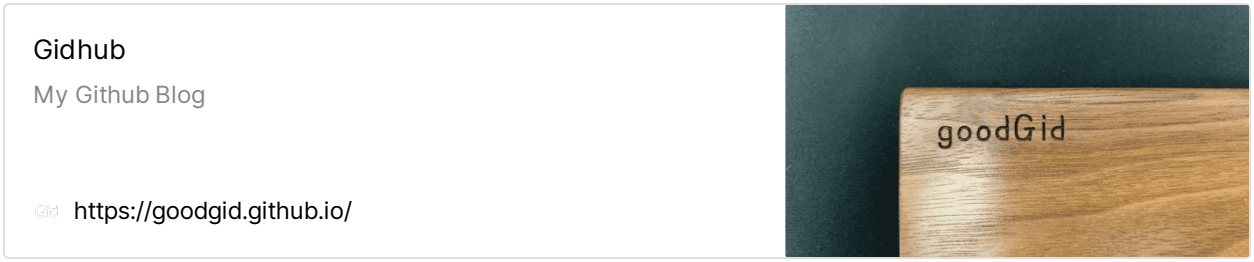
스프링 MVC 모킹하기

- 테스트에서 Mock MVC를 설정하려면 MockMvcBuilders를 사용한다. 이 클래스는 정적 메서드 두 개를 제공한다.
- standaloneSetup() : 수동으로 생성하고 구성한 컨트롤러 한 개 이상을 서비스할 Mock MVC를 만든다.
- webAppContextSetup() : 구성된 컨트롤러 한 개 이상을 포함하는 스프링 애플리케이션 컨텍스트를 사용하여 Mock MVC를 만든다.
-

이 두 옵션의 가장 큰 차이는 standaloneSetup() 메서드는 테스트할 컨트롤러를 수동으로 초기화하고 주입하기를 기대하는 반면, webAppContextSetup() 메서드는 (스프링이 로드한) WebApplicationContext의 인스턴스로 작동한다는 점이다.

standaloneSetup() 메서드는 한 컨트롤러에 집중하여 테스트하는 용도로만 사용한다는 점에서 유닛 테스트와 유사하다. 반면에 webAppContextSetup() 메서드는 스프링이 컨트롤러는 물론 의존성까지 로드하여 완전한 통합 테스트를 할 수 있게 한다.

여기서는 webAppContextSetup() 메서드를 사용하여 스프링 부트가 자동으로 구성한 애플리케이션 컨텍스트에서 초기화하여 주입된 것처럼 BoardController를 테스트하자.



유닛 테스트란?

- 단위 테스트(Unit test)는 프로그램의 기본 단위인 모듈을 테스트하여 모듈 테스트라고도 한다.
- 구현 단계에서 각 모듈의 개발을 완료한 후 개발자가 명세서의 내용대로 정확히 구현되었는지를 테스트한다.
- 즉 개별 모듈이 제대로 구현되어 정해진 기능을 정확히 수행하는지를 테스트한다.
- 화이트박스 테스트와 블랙박스 테스트를 모두 사용할 수 있지만 모듈 내부의 구조를 구체적으로 들여다 볼 수 있는 화이트박스 테스트같은 구조적 테스트를 주로 시행한다.
- 단위 테스트가 개발된 모듈만 테스트하기 때문에 쉬울 것 같지만 시스템은 수 많은 모듈이 서로 정보를 주고 받으며 연결되어 있다.
- 즉 테스트할 모듈을 호출하는 모듈도 있고 테스트할 모듈이 호출하는 모듈도 있다.
- 따라서 한 모듈을 테스트하려면 그 모듈과 직접 관련된 상위 모듈과 하위 모듈까지 모두 존재해야 정확히 테스트할 수 있다.
- 그러나 하나의 모듈을 테스트할 때 상위나 하위 모듈이 개발이 안 된 경우도 있다.
- 이때 상위나 하위 모듈이 개발될 때 까지 기다릴 수 는 없으므로 가상의 상위나 하위 모듈을 만들어 사용해야 한다.
- 상위 모듈의 역할을 하는 가상의 모듈을 테스트 드라이버라 하고 그 역할은 테스트할 모듈을 호출하는 것이다.

- 즉, 필요한 데이터를 인자를 통하여 넘겨주고 테스트가 완료된 후 그 결과 값을 받는 역할을 해준다.
- 반대로 하위 모듈의 역할을 하는 모듈을 테스트 스텁(stub)이라고 한다.
- 스텁 모듈은 테스트 할 모듈이 호출할 때 인자를 통해 받은 값을 가지고 수행한 후 그 결과를 테스트할 모듈에 넘겨주는 역할을 한다.
- 따라서 드라이버와 스텁 모듈은 테스트할 때 필요한 기능만 제공할 있도록 단순히 구현한다.

Unit Test의 장점

1. 문제점 발견
    - 유닛 테스트의 목적은 프로그램의 각 부분을 고립 시켜서 각각의 부분이 정확하게 동작하는지 확인하는 것이다.
    - 즉, 프로그램을 작은 단위로 쪼개서 각 단위가 정확하게 동작하는지 검사하고, 이를 통해 문제 발생 시 정확하게 어느 부분이 잘못되었는지를 재빨리 확인할 수 있게 해준다.
    - 따라서 프로그램의 안정성이 높아진다. 유닛 테스트는 일견 개발 시간을 증가 시키는 것처럼 보이지만 개발 기간 중 대부분을 차지하는 디버깅 시간을 단축시킴으로써 여유로운 프로그래밍을 가능케 한다.

2. 변경이 쉽다.
    - 프로그래머는 언제라도 유닛 테스트를 믿고 리팩토링을 할 수 있다. 리팩토링 후에도 해당 모듈이 의도대로 작동하고 있음을 유닛 테스트를 통해서 확신할 수 있다. 이를 회귀 테스트(Regression testing)라 한다.
    - 어떻게 코드를 고치더라도 문제점을 금방 파악할 수 있고 수정된 코드가 정확하게 동작하는지 쉽게 알 수 있게 되므로 프로그래머들은 더욱 더 의욕적으로 코드를 변경할 수 있게 된다.
    - 좋은 유닛 테스트 디자인은 그 유닛이 사용되는 모든 경로를 커버할 수 있는 테스트 케이스를 만들어 준다.

3. 통합이 간단하다.

- 유닛 테스트는 유닛 자체의 불확실성을 제거해주므로 상향식(bottom-up) 테스트 방식에서 유용하다.
- 먼저 프로그램의 각 부분을 검증하고 그 부분들은 합쳐서 다시 검증하는 통합 테스트에서 더욱 더 빛을 발한다.