[Spring Boot] Test Code

테스트 코드를 작성해야 하는 이유

- 개발단계 초기에 문제를 발견할 수 있음
- 나중에 코드를 리팩토링하거나 라이브러리 업그레이드 시 기존 기능이 잘 작동하는 지 확인 가능함
- 기능에 대한 불확실성 감소

개발 코드 이외에 테스트 코드를 작성하는 일은 개발 시간이 늘어날 것이라고 생각할 수 있다. 하지만 내 코드에 오류가 있는 지 검증할 때, 테스트 코드를 작성하지 않고 진행한다면 더 시간 소모가 클 것이다.

```
1. 코드를 작성한 뒤 프로그램을 실행하여 서버를 킨다.
2. API 프로그램(ex. Postman)으로 HTTP 요청 후 결과를 Print로 찍어서 확인한다.
3. 결과가 예상과 다르면, 다시 프로그램을 종료한 뒤 코드를 수정하고 반복한다.
```

위와 같은 방식이 얼마나 반복될 지 모른다. 그리고 하나의 기능마다 저렇게 테스트를 하면 서버를 키고 끄는 작업 또한 너무 비효율적이다.

이 밖에도 Print로 눈으로 검증하는 것도 어느정도 선에서 한계가 있다. 테스트 코드는 자동으로 검증을 해주기 때문에 성공한다면 수동으로 검증할 필요 자체가 없어진다.

새로운 기능이 추가되었을 때도 테스트 코드를 통해 만약 기존의 코드에 영향이 갔다면 어떤 부분을 수정해야 하는 지 알 수 있는 장점도 존재한다.

따라서 테스트 코드는 개발하는 데 있어서 필수적인 부분이며 반드시 활용해야 한다.

테스트 코드 예제

```
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;
import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;

@RunWith(SpringRunner.class)
```

```
@WebMvcTest(controllers = HomeController.class)
public class HomeControllerTest {

    @Autowired
    private MockMvc mvc;

    @Test
    public void home_return() throws Exception {
        //when
        String home = "home";

        //then
        mvc.perform(get("/home"))
                .andExpect(status().isOk())
                .andExpect(content().string(home));
    }
}
```

1. `@RunWith(SpringRunner.class)`

테스트를 진행할 때 JUnit에 내장된 실행자 외에 다른 실행자를 실행시킨다.

스프링 부트 테스트와 JUnit 사이의 연결자 역할을 한다고 생각하면 된다.

2. `@WebMvcTest`

컨트롤러만 사용할 때 선언이 가능하며, Spring MVC에 집중할 수 있는 어노테이션이다.

3. `@Autowired`

스프링이 관리하는 Bean을 주입시켜준다.

4. `MockMvc`

웹 API를 테스트할 때 사용하며, 이를 통해 HTTP GET, POST, DELETE 등에 대한 API 테스트가 가능하다.

5. `mvc.perform(get("/home"))`

`/home` 주소로 HTTP GET 요청을 한 상황이다.

6. `.andExpect(status().isOk())`

결과를 검증하는 `andExpect`로, 여러개를 붙여서 사용이 가능하다. `status()`는 HTTP Header를 검증하는 것으로 결과에 대한 HTTP Status 상태를 확인할 수 있다. 현재 `isOK()`는 200 코드가 맞는지 확인하고 있다.

프로젝트를 만들면서 다양한 기능들을 구현하게 되는데, 이처럼 테스트 코드로 견고한 프로젝트를 만들기 위한 기능별 단위 테스트를 진행하는 습관을 길러야 한다.

[참고 자료]

- 링크