스프링부트로 **RestFulAPI** 구현하기

5장 테스트코드

- **JUnit**
- **AssertJ**
- **@Test**
- **given when then**

박명회

5장 JUnit, AssertJ

JUnit은 애플리케이션의 코드 품질을 높이고, 버그를 조기에 발견하여 안정적인 소프트웨어를 개발하는 데 중요한 역할을 합니다.

AssertJ는 코드의 가독성을 높이고, 풍부한 어설션 메서드를 제공하여 단위 테스트를 더 간결하고 명확하게 작성할 수 있게 합니다.

테스트 코드의 한 방식인 **given-when-then** 패턴을 많이 사용 하고 있습니다.

given은 테스트 실행을 준비하는 단계

when은 테스트를 진행하는 단계

then은 테스트 결과를 검증하는 단계입니다.

5장 JUnit Assertj를 활용한 기본 테스트 코드

```
@DisplayName("1 + 2는 3이다")
@Test
public void basicMathTest() {
    int a = 1;
    int b = 2;
    int sum = 3;

    Assertions.assertEquals(a + b, sum);
}
```

5장 AssertJ 메서드

```
import org.assertj.core.api.Assertions;
import org.junit.jupiter.api.Test;

import java.util.Arrays;
import java.util.List;

public class AssertJExamples {

    @Test
    public void testBasicAssertions() {
        int a = 1;
        int b = 2;
        int sum = 3;

        Assertions.assertThat(a + b).isEqualTo(sum);
        Assertions.assertThat(a).isNotEqualTo(b);
    }

    @Test
    public void testStringAssertions() {
        String str = "Hello, World!";
```

5장 AssertJ 메서드

메서드	설명
`isEqualTo(expected)`	두 값이 같은지 확인합니다.
`isNotEqualTo(notExpected)`	두 값이 같지 않은지 확인합니다.
`isNull()`	값이 `null`인지 확인합니다.
`isNotNull()`	값이 `null`이 아닌지 확인합니다.
`contains(substring)`	문자열이나 컬렉션에 특정 요소가 포함되어 있는지 확인합니다.
`doesNotContain(element)`	문자열이나 컬렉션에 특정 요소가 포함되지 않았는지 확인합니다.
`hasSize(expectedSize)`	컬렉션의 크기를 확인합니다.
`isInstanceOf(exceptionClass)`	예외의 타입을 확인합니다.
`hasMessageContaining(message)`	예외 메시지가 특정 문자열을 포함하는지 확인합니다.

2장 given when then

```
public class CollectionAssertionsTest {

    @Test
    public void testCollectionAssertions() {

        // Given: 이름 리스트가 주어졌을 때
        List<String> names = Arrays.asList("John", "Jane", "Jack");

        // When: 리스트를 검증할 때
        // Then:
       // 리스트는 3개의 요소를 가지고 있고,
       // "John"과 "Jane"을 포함하며, "Joe"를 포함하지 않아야 한다
        Assertions.assertThat(names)
            .hasSize(3)
            .contains("John", "Jane")
            .doesNotContain("Joe");

    }
}
```

5장 기본 테스트 라이프 싸이클

@Test : 테스트 메서드임을 나타냅니다.

@BeforeEach : 각 테스트 메서드가 실행되기 전에 실행됩니다.

@AfterEach : 각 테스트 메서드가 실행된 후에 실행됩니다.

@BeforeAll : 모든 테스트 메서드가 실행되기 전에 한 번 실행됩니다.

@AfterAll : 모든 테스트 메서드가 실행된 후에 한 번 실행됩니다.

@DisplayName : 테스트 메서드의 설명을 지정할 수 있습니다.

@Disabled : 테스트 메서드를 비활성화합니다.

2장 @Configuration과 @Bean

```
import org.junit.jupiter.api.*;

public class AnnotationExampleTest {

    @BeforeAll
    public static void setupBeforeAll() {
        System.out.println("BeforeAll: 모든 테스트 메서드가 실행되기 전에 한 번 실행됩니다.");
    }

    @BeforeEach
    public void setupBeforeEach() {
        System.out.println("BeforeEach: 각 테스트 메서드가 실행되기 전에 실행됩니다.");
    }

    @Test
    @DisplayName("Test 1: 단순한 테스트 메서드")
    public void testMethod1() {
        System.out.println("Test 1: 테스트 메서드임을 나타냅니다.");
        Assertions.assertTrue(true);
    }
```

5장 **UserController** 테스트 코드 만들기

```
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.ResultActions;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.web.context.WebApplicationContext;

import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
```

5장 UserController 테스트 코드 만들기

```
@SpringBootTest
@AutoConfigureMockMvc
class UserControllerTest {

    @Autowired
    protected MockMvc mockMvc;

    @Autowired
    private WebApplicationContext context;

    @Autowired
    private UserRepository userRepository;
```

MockMvc를 사용하여 **HTTP** 요청을 모의로 만들어 컨트롤러 메서드를 호출하고, 응답을 검증합니다.

WebApplicationContext는 **Spring**의 애플리케이션 컨텍스트 중 하나로, 웹 애플리케이션과 관련된 모든 빈을 포함합니다.
테스트 시 **MockMvc**를 설정할 때 사용됩니다. **MockMvcBuilders.webAppContextSetup(context)**를 통해 **MockMvc**가 애플리케이션 컨텍스트를 사용하도록 설정할 수 있습니다.

UserRepository는 **Spring Data JPA**에서 제공하는 레포지토리 인터페이스로, 데이터베이스와의 **CRUD** 작업을 처리합니다.

5장 UserController 테스트 코드 만들기

```
@BeforeEach
public void mockMvcSetUp() {
    this.mockMvc = MockMvcBuilders.webAppContextSetup(context)
            .build();
}

@AfterEach
public void cleanUp() {
    userRepository.deleteAll();
}
```

MockMvcBuilders.webAppContextSetup(context) : **MockMvc**를 **WebApplicationContext**로 설정합니다. 이를 통해 **MockMvc**가 **Spring** 애플리케이션 컨텍스트에 접근하여 컨트롤러와 그 외 빈들을 사용할 수 있게 합니다.

this.mockMvc = ... : 설정된 **MockMvc** 객체를 현재 클래스의 **mockMvc** 필드에 할당합니다. 이렇게 함으로써 각 테스트 메서드가 실행되기 전에 항상 새로운 **MockMvc** 객체가 생성되고 설정됩니다.

@BeforeEach 와 **@AfterEach**를 사용함으로써 각 테스트 코드 실행전과 실행후에 실행하게 됩니다.

5장 UserController 테스트 코드 만들기

```
@DisplayName("getAllUsers: 모든 사용자 조회에 성공한다.")
@Test
public void getAllUsers() throws Exception {
    // given
    final String url = "/users";
    User savedUser = userRepository.save(new User(1L, "John Doe"));

    // when
    final ResultActions result = mockMvc.perform(get(url)
            .accept(MediaType.APPLICATION_JSON));

    // then
    result
            .andExpect(status().isOk())
            .andExpect(jsonPath("$[0].id").value(savedUser.getId()))
            .andExpect(jsonPath("$[0].name").value(savedUser.getName()));
}
```

5장 UserController 테스트 코드 만들기

```
@DisplayName("createUser: 새로운 사용자를 생성한다.")
@Test
public void createUser() throws Exception {
    // given
    final String url = "/users";
    final String userJson = "{\"id\": 2, \"name\": \"Jane Doe\"}";

    // when
    final ResultActions result = mockMvc.perform(post(url)
            .content(userJson)
            .contentType(MediaType.APPLICATION_JSON)
            .accept(MediaType.APPLICATION_JSON));

    // then
    result
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.id").value(2))
            .andExpect(jsonPath("$.name").value("Jane Doe"));
}
```

5장 UserController 테스트 코드 만들기

```
@Test
public void objectMapperTest(){
    User user = new User(2L, "Jane Doe");
    ObjectMapper objectMapper = new ObjectMapper();
    String userJson = objectMapper.writeValueAsString(user);
    System.out.println(userJson);
}
```

감사합니다