УДК 004

Написание модульного теста с использованием Spring Boot

Еровлев Павел Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Еровлева Регина Викторовна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье рассматривается как написать модульный тест для REST приложения. В статье используются IntelliJ IDEA, SpringBoot и язык программирования Java. Конечным результатом являются созданные и успешно пройденные тесты.
Ключевые слова: Java, Rest, Spring

Writing a unit test using Spring Boot

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Erovleva Regina Victorovna
Sholom-Aleichem Priamursky State University
Student

Abstract
This article covers how to write a unit test for a REST application. This article uses IntelliJ IDEA, SpringBoot, and the Java programming language. The end result is created and successfully passed tests.
Keywords: Java, Rest, Spring

1 Введение

1.1 Актуальность

Тестирование это один из самых актуальных решений в программировании. Хоть само тестирование и не может обеспечить полной уверенности в безошибочной работе продукта, но оно достаточно выручает при проверки отдельных модулей. Чем меньше ошибок и проблем с работой в отдельных модулях, тем выше вероятность, что при запуске продукт не сломается в первое же время от высоких нагрузок или случайно оставленном в коде ненужном символе.

1.2 Обзор исследований

А.Н. Иванов и П.С. Власюк описали в своей работе реализацию безопасного веб-приложения на языке программирования Java [1]. М.А. Потовиченко, М.В. Привалов и С.В. Корнев рассмотрели разработку программного продукта, который обеспечивает учет данных посещения занятий студентов, а также защиту их работ [2]. В.А. Сухомлин описал кратко в своей статье принцип работа комплексной программы дополнительного образования, ориентированную на подготовку разработчиков Java enterprise приложений [3]. Д.В. Козырев, Л.А. Володченкова разработали программный интерфейс серверной части для облачного хранилища данных [4]. Ю.А. Флёров и Л.Л. Вышинский разработали программу для организации взаимодействия с Web-клиентами в программных комплексах [5].

1.3 Цель исследования

Цель исследования – создать модульные тесты для проверки корректности работы REST сервиса.

2 Материалы и методы

Для реализации будет использоваться язык программирования Java, IntelliJ IDEA и библеотека Junit.

3 Результаты и обсуждения

Spring Boot упрощает модульное тестирование с помощью «SpringRunner». Также проще писать модульные тесты для «REST Controller» с «MockMVC».

Для написания модульного теста необходимо добавить зависимость «Spring Boot Starter Test» в файл конфигурации сборки.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

Рисунок 1 – Добавление зависимости

Реализация модульного теста будет происходить для класса StudentService.java, который реализует методы обращения к бд.

```java
public interface StudentService {

    3 usages  1 implementation
    List<Student> findAll();

    5 usages  1 implementation
    Student findByStudentNumber(long studentNumber);

    2 usages  1 implementation
    Student findByEmail(String email);

    3 usages  1 implementation
    List<Student> findAllByOrderByGpaDesc();

    3 usages  1 implementation
    Student saveOrUpdateStudent(Student student);

    3 usages  1 implementation
    void deleteStudentById(String id);

}
```

Рисунок 2 – StudentService.java

Для создания модульного теста необходимо будет, во-первых, написать аннотацию «@RunWith» над классом. Эта аннотация используется «SpringJUnit4ClassRunner» для запуска модульных тестов.

Во-вторых, добавить аннотацию «@SpringBootTest». Эта аннотация предоставляет множество возможностей для тестовых классов.

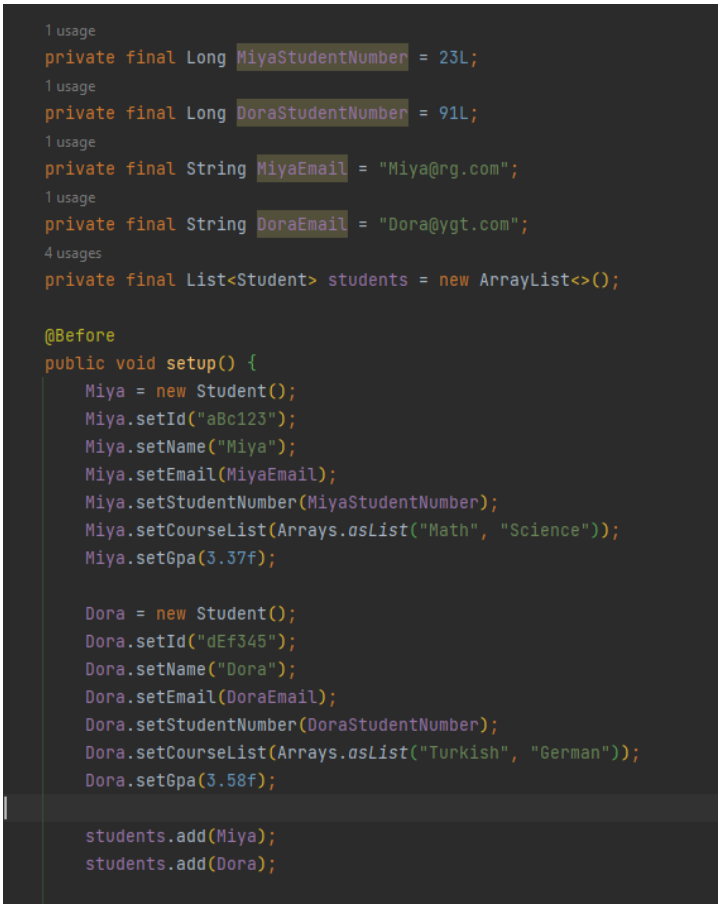
Затем добавляется тестируемый класс «StudentService.java» и его зависимости.

```java
@RunWith(SpringRunner.class)
public class StudentServiceTest {
    6 usages
    @Autowired
    private StudentService studentService;

    6 usages
    @MockBean
    private StudentRepository studentRepository;
```

Рисунок 3 – Создание модульного теста

Теперь можно написать фиктивные объекты для тестирования «StudentService.java». Создадим студентов Dora и Miya. Объектам присвоим следующие значения.



```
1 usage
private final Long MiyaStudentNumber = 23L;
1 usage
private final Long DoraStudentNumber = 91L;
1 usage
private final String MiyaEmail = "Miya@rg.com";
1 usage
private final String DoraEmail = "Dora@ygt.com";
4 usages
private final List<Student> students = new ArrayList<>();

@Before
public void setup() {
    Miya = new Student();
    Miya.setId("aBc123");
    Miya.setName("Miya");
    Miya.setEmail(MiyaEmail);
    Miya.setStudentNumber(MiyaStudentNumber);
    Miya.setCourseList(Arrays.asList("Math", "Science"));
    Miya.setGpa(3.37f);

    Dora = new Student();
    Dora.setId("dEf345");
    Dora.setName("Dora");
    Dora.setEmail(DoraEmail);
    Dora.setStudentNumber(DoraStudentNumber);
    Dora.setCourseList(Arrays.asList("Turkish", "German"));
    Dora.setGpa(3.58f);

    students.add(Miya);
    students.add(Dora);
```

Рисунок 4 – Добавление фиктивных пользователей

Для метода «findAll()» в «StudentService» нужно имитировать метод «findAll()» в «StudentRepository.java». После этого можно написать тестовый метод. Для этого написан метод «testFindAll_thenStudentListShouldBeReturned()». Этот метод просто проверяет размер возвращаемых данных.

```
    Mockito.when(studentRepository.findAll()).thenReturn(students);

    Mockito.when(studentRepository.findByStudentNumber(MiyaStudentNumber))
            .thenReturn(Miya);

    Mockito.when(studentRepository.findByEmail(DoraEmail))
            .thenReturn(Dora);

    Mockito.when(studentRepository.findAllByOrderByGpaDesc())
            .thenReturn(students.stream().sorted(
                    Comparator.comparing(Student::getGpa).reversed()).collect(Collectors.toList()));

    Mockito.when(studentRepository.save(Miya)).thenReturn(Miya);
}

@Test
public void testFindAll_thenStudentListShouldBeReturned() {
    List<Student> foundStudents = studentService.findAll();

    assertNotNull(foundStudents);
    assertEquals( expected: 2, foundStudents.size());
}
```

Рисунок 5 – Метод проверки введенных данных

Для метода «findByStudentNumber()» в «StudentService» нужно имитировать «findByStudentNumber()» метод в «StudentRepository.java». После этого можно написать тестовый метод. Для этого написан метод «testFindByStudentNumber23_thenRagcrixShouldBeReturned()».

```
@Test
public void testFindByStudentNumber23_thenRagcrixShouldBeReturned() {
    Student found = studentService.findByStudentNumber(MiyaStudentNumber);

    assertNotNull(found);
    assertEquals(Miya.getName(), found.getName());
    assertEquals(Miya.getId(), found.getId());
}
```

Рисунок 6 – Метод проверки введеного номера

Для методов «findByEmail()», «findAllByOrderByGpaDesc()» и «saveOrUpdateStudent()», в классе «StudentService» в методе «deleteStudentById()» связанные методы в «StudentRepository.java» имитируются одинаково. Затем записываются методы тестов для «StudentService».

```
    @Test
    public void testFindByEmail_thenYigitShouldBeReturned() {
        Student found = studentService.findByEmail(DoraEmail);

        assertNotNull(found);
        assertEquals(Dora.getName(), found.getName());
        assertEquals(Dora.getId(), found.getId());
    }

    @Test
    public void testFindAllByOrderByGpaDesc_thenStudentsShouldBeReturned_byGPADesc() {
        List<Student> foundStudents = studentService.findAllByOrderByGpaDesc();

        assertNotNull(foundStudents);
        assertEquals( expected: 2, foundStudents.size());
        assertEquals(Dora.getName(), foundStudents.get(0).getName());
        assertEquals(Dora.getId(), foundStudents.get(0).getId());
    }

    @Test
    public void testSaveOrUpdateStudent_thenStudentShouldBeReturned() {
        Student found = studentService.saveOrUpdateStudent(Miya);

        assertNotNull(found);
        assertEquals(Miya.getName(), found.getName());
        assertEquals(Miya.getId(), found.getId());
    }

    @Test
    public void testDeleteStudentById() {
        studentService.deleteStudentById(Miya.getId());

        Mockito.verify(studentRepository, Mockito.times( wantedNumberOfInvocations: 1))
                .deleteById(Miya.getId());
    }

}
```

Рисунок 7 – Методы проверки введенных значений

Теперь запустим тесты и проверим как они отработают.

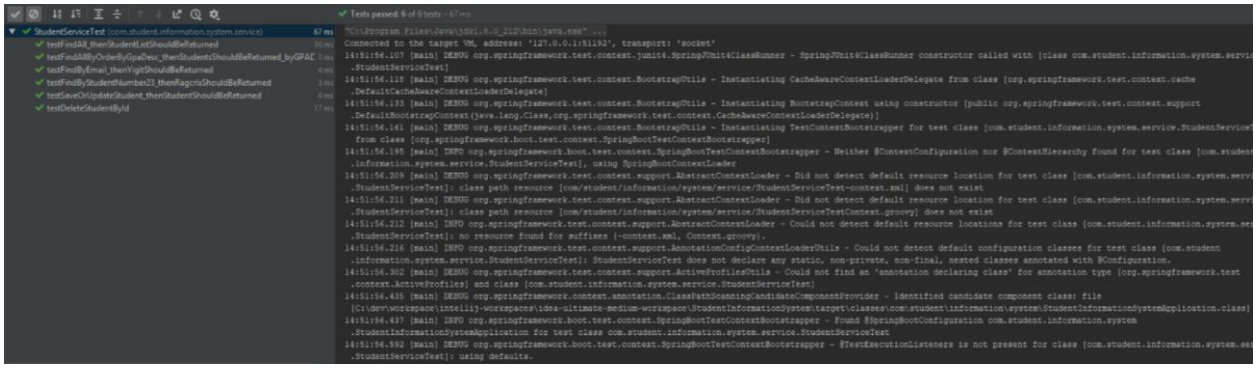


Рисунок 8 – Результат тестирования

Так как в введенных значениях были ошибки, следовательно тестирование прошло успешно и сделан вывод, что введенные значения для студентов не являются правильными. После обработки полученных сообщений от тестирования, можно закрыть проблему не правильно введенных данных, на пример установить дополнительную проверку вводимых значений на этапе отправки в бд, и если данные введены неправильно, то не производить запись в бд и сообщить пользователю о неправильности данных.

Выводы

В данной статье был рассмотрен процесс создания модульных тестов с использованием языка программирования Java, фреймворка SpringBoot и библиотеки Junit.

Библиографический список

1. Иванов А.Н., Власюк П.С. Разработка приложения с использованием веб-технологий java // Тихоокеанский государственный университет. 2020. С. 261-266.
2. Потовиченко М.А., Привалов М.В., Корнев С.В. Компьютеризированная подсистема учета текущей успеваемости студента в условиях вуза // Информатика, управляющие системы, математическое и компьютерное моделирование. 2019. № 2. С. 71-75.
3. Сухомлин В.А. Подготовка разработчиков корпоративных java-приложений в режиме дистанционного обучения // Информатика, управляющие системы, математическое и компьютерное моделирование. 2013. № 9. С. 175-180.
4. Володченкова Л.А., Козырев Д.В. Разработка серверной части программного приложения для удаленного хранения данных// Математические структуры и моделирование. 2020. № 1 (53). С. 108-138.
5. Флёров Ю.А. и Вышинский Л.Л. Программа организации интерфейса web-серверов с java-приложениями // Аллея науки. 2016. № 7. С. 58-61.