УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Рефакторинг баз данных и приложений»

Этап №3

Рефакторинг существующего проекта

Студенты

Белогаев Д. В.

Кузнецов М. А.

P34131

Преподаватель

Логинов И. П.

Санкт-Петербург, 2023 г.

Задание

Провести рефакторинг приложения, в качестве приложения была взята парная лабораторная работа по дисциплине БЛПС, приведенная к 3-ему этапу (так как 4-й этап полностью меняет логику предыдущих этапов и не содержит сложной логики).

Выполнение работы

В рамках третьего этапа были написаны тесты по оба сервиса Producer и Consumer:

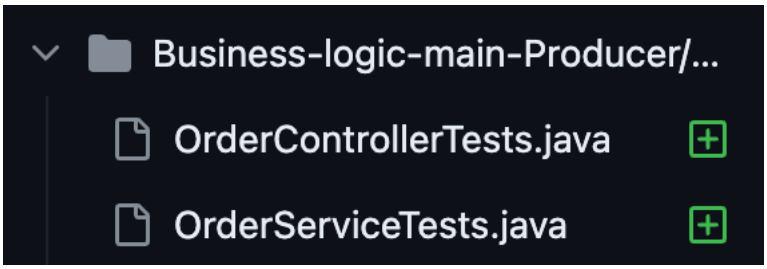
1. Использовали Junit для написания тестов
2. Использовали Mockito для мока необходимых компонентов

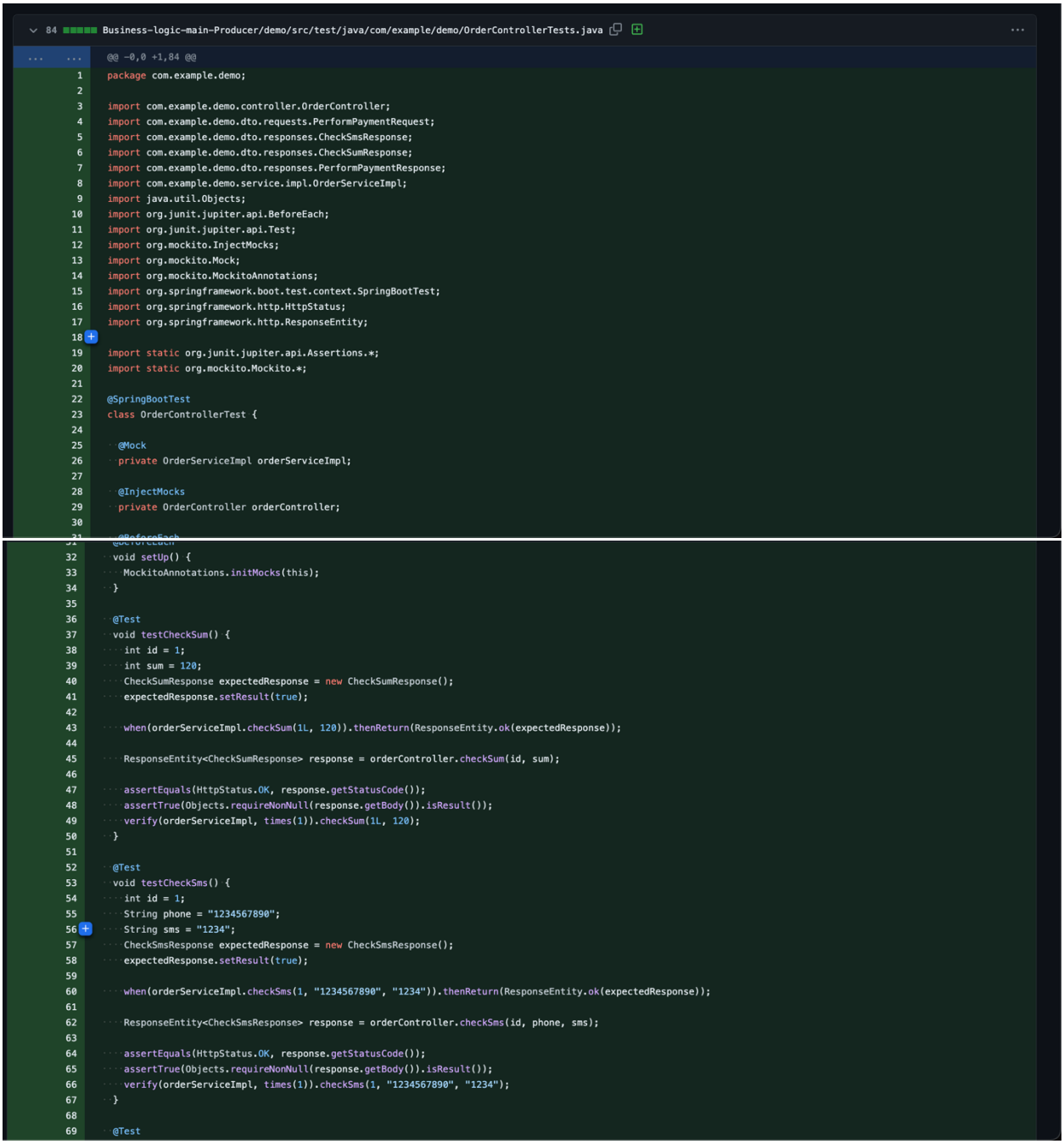
Ссылка на коммит:

Для Producer: https://github.com/Icerzack/blps-ref/commit/e3a3a5c7ffe8b02b279b1110ded209d8da231cbf

Для Consumer: https://github.com/Icerzack/blps-ref/commit/2425cbc33b09640d36021b8c6fdbd39d69f9ffeb

Producer:





84 Business-logic-main-Producer/demo/src/test/java/com/example/demo/OrderControllerTests.java

```
@@ -0,0 +1,84 @@
package com.example.demo;

import com.example.demo.controller.OrderController;
import com.example.demo.dto.requests.PerformPaymentRequest;
import com.example.demo.dto.responses.CheckSmsResponse;
import com.example.demo.dto.responses.CheckSumResponse;
import com.example.demo.dto.responses.PerformPaymentResponse;
import com.example.demo.service.impl.OrderServiceImpl;
import java.util.Objects;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

@SpringBootTest
class OrderControllerTest {

    @Mock
    private OrderServiceImpl orderServiceImpl;

    @InjectMocks
    private OrderController orderController;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.initMocks(this);
    }

    @Test
    void testCheckSum() {
        int id = 1;
        int sum = 120;
        CheckSumResponse expectedResponse = new CheckSumResponse();
        expectedResponse.setResult(true);

        when(orderServiceImpl.checkSum(1L, 120)).thenReturn(ResponseEntity.ok(expectedResponse));

        ResponseEntity<CheckSumResponse> response = orderController.checkSum(id, sum);

        assertEquals(HttpStatus.OK, response.getStatusCode());
        assertTrue(Objects.requireNonNull(response.getBody()).isResult());
        verify(orderServiceImpl, times(1)).checkSum(1L, 120);
    }

    @Test
    void testCheckSms() {
        int id = 1;
        String phone = "1234567890";
        String sms = "1234";
        CheckSmsResponse expectedResponse = new CheckSmsResponse();
        expectedResponse.setResult(true);

        when(orderServiceImpl.checkSms(1, "1234567890", "1234")).thenReturn(ResponseEntity.ok(expectedResponse));

        ResponseEntity<CheckSmsResponse> response = orderController.checkSms(id, phone, sms);

        assertEquals(HttpStatus.OK, response.getStatusCode());
        assertTrue(Objects.requireNonNull(response.getBody()).isResult());
        verify(orderServiceImpl, times(1)).checkSms(1, "1234567890", "1234");
    }

    @Test
```

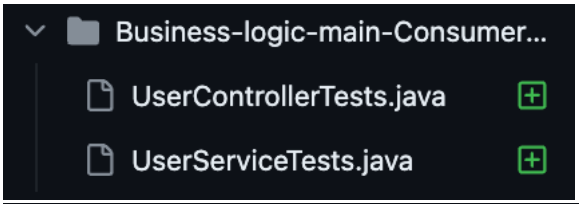
```
@Test
void testPerformPayment() {
    PerformPaymentRequest request = new PerformPaymentRequest(1L, "1234567890123456", "12/25", "123", 100.0, "Address");
    PerformPaymentResponse expectedResponse = new PerformPaymentResponse();
    expectedResponse.setResult(true);

    when(orderServiceImpl.performPayment(1L, "1234567890123456", "12/25", "123", 100.0, "Address"))
            .thenReturn(ResponseEntity.ok(expectedResponse));

    ResponseEntity<PerformPaymentResponse> response = orderController.performPayment(request);

    assertEquals(HttpStatus.OK, response.getStatusCode());
    assertTrue(Objects.requireNonNull(response.getBody()).isResult());
    verify(orderServiceImpl, times(1)).performPayment(1L, "1234567890123456", "12/25", "123", 100.0, "Address");
}
}
```

104 Business-logic-main-Producer/demo/src/test/java/com/example/demo/OrderServiceTests.java

```
@@ -0,0 +1,104 @@
package com.example.demo;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.ArgumentMatchers.eq;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.verifyNoInteractions;

import com.example.demo.dto.requests.PerformPaymentRequest;
import com.example.demo.dto.responses.CheckSmsResponse;
import com.example.demo.dto.responses.CheckSumResponse;
import com.example.demo.dto.responses.PerformPaymentResponse;
import com.example.demo.service.impl.OrderServiceImpl;
import java.util.Objects;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
@SpringBootTest
public class OrderServiceTests {

    @Mock
    private KafkaTemplate<String, PerformPaymentRequest> kafkaTemplate;

    @InjectMocks
    private OrderServiceImpl orderService;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.initMocks(this);
    }

    @Test
    void testCheckSum() {
        ResponseEntity<CheckSumResponse> response = orderService.checkSum(123L, 150.0);
        assertTrue(Objects.requireNonNull(response.getBody()).isResult());
    }

    @Test
    void testCheckSms() {
        ResponseEntity<CheckSmsResponse> response = orderService.checkSms(123, "1234567890", "1234");
        assertFalse(Objects.requireNonNull(response.getBody()).isResult());
    }

    @Test
    void testPerformPaymentValid() {
        ResponseEntity<PerformPaymentResponse> response = orderService.performPayment(1L, "1234567890123456",
                "12/25", "123", 500.0, "123 Street");
        assertTrue(Objects.requireNonNull(response.getBody()).isResult());
        verify(kafkaTemplate, times(1)).send(eq("perform-test3"), any(PerformPaymentRequest.class));
    }

    @Test
    void testPerformPaymentInvalid() {
        ResponseEntity<PerformPaymentResponse> response = orderService.performPayment(1L, "123",
                "12/25", "123", 500.0, "123 Street");
        assertFalse(Objects.requireNonNull(response.getBody()).isResult());
```

```java
    @Test
    void testPerformPaymentWithInvalidCardCVV() {
        ResponseEntity<PerformPaymentResponse> response = orderService.performPayment(
                1L, "1234567890123456", "12/25", "12", 500.0, "123 Street");

        assertFalse(Objects.requireNonNull(response.getBody()).isResult());
        verifyNoInteractions(kafkaTemplate);
    }

    @Test
    void testPerformPaymentWithInvalidCardNum() {
        ResponseEntity<PerformPaymentResponse> response = orderService.performPayment(
                1L, "123", "12/25", "123", 500.0, "123 Street");

        assertFalse(Objects.requireNonNull(response.getBody()).isResult());
        verifyNoInteractions(kafkaTemplate);
    }

    @Test
    void testPerformPaymentWithInvalidCost() {
        ResponseEntity<PerformPaymentResponse> response = orderService.performPayment(
                1L, "1234567890123456", "12/25", "123", -50.0, "123 Street");

        assertFalse(Objects.requireNonNull(response.getBody()).isResult());
        verifyNoInteractions(kafkaTemplate);
    }

    @Test
    void testPerformPaymentWithValidPaymentDetails() {
        ResponseEntity<PerformPaymentResponse> response = orderService.performPayment(
                1L, "1234567890123456", "12/25", "123", 500.0, "123 Street");

        assertTrue(Objects.requireNonNull(response.getBody()).isResult());
        verify(kafkaTemplate, times(1)).send(eq("perform-test3"), any(PerformPaymentRequest.class));
    }

}
```

Consumer:



Business-logic-main-Consumer/demo/src/test/java/com/example/demo/UserControllerTests.java

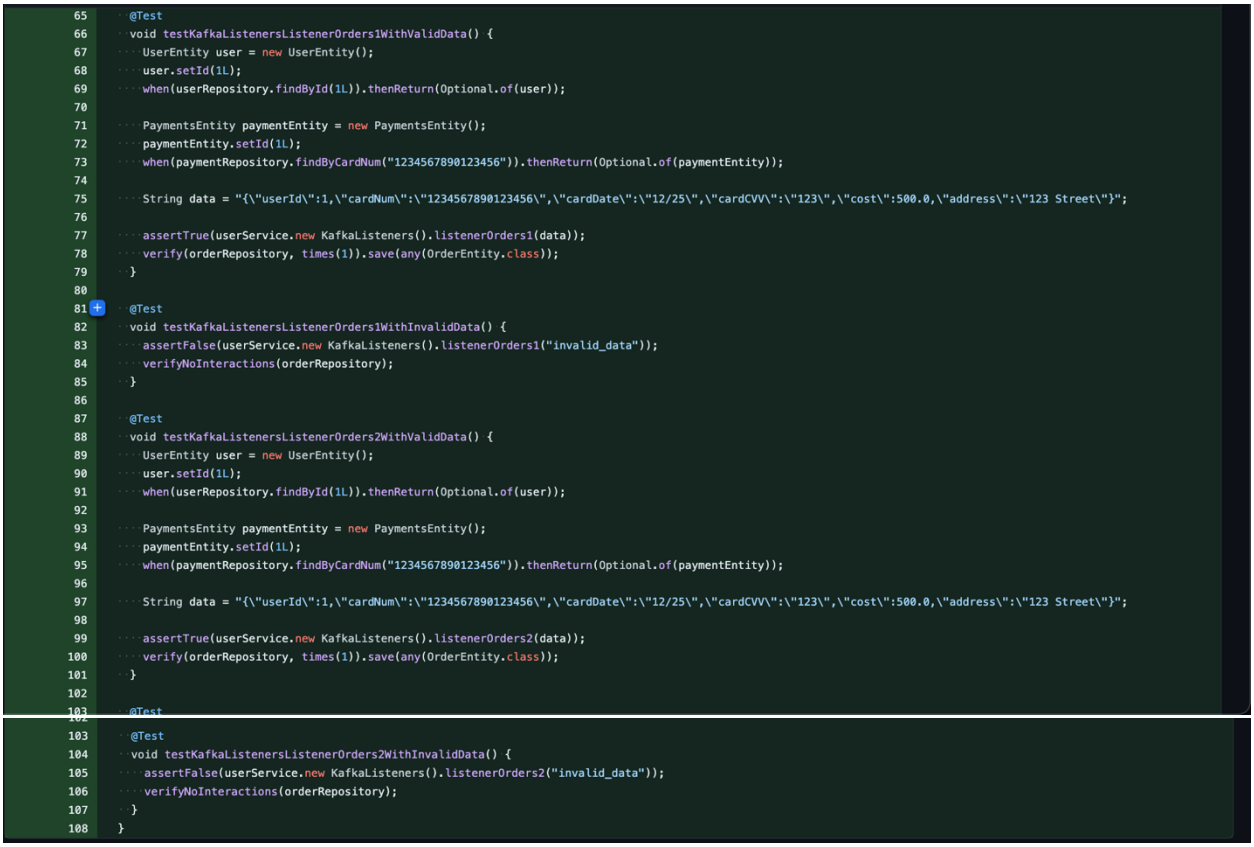
```java
import com.example.demo.dto.requests.AddPaymentRequest;
import com.example.demo.dto.requests.AddPhoneRequest;
import com.example.demo.dto.responses.*;
import com.example.demo.service.impl.UserServiceImpl;
import java.util.Objects;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

@SpringBootTest
class UserControllerTests {

    @Mock
    private UserServiceImpl userServiceImpl;

    @InjectMocks
    private UserController userController;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
    }

    @Test
    void testAddPhone() {
        AddPhoneRequest request = new AddPhoneRequest();
        request.setUserId(1L);
        request.setPhoneNumber("1234567890");
        AddPhoneResponse expectedResponse = new AddPhoneResponse();
        expectedResponse.setResult(true);
```

```
43      when(userServiceImpl.addPhone(1L, "1234567890")).thenReturn(ResponseEntity.ok(expectedResponse));
44
45      ResponseEntity<AddPhoneResponse> response = userController.addPhone(request);
46
47      assertEquals(HttpStatus.OK, response.getStatusCode());
48      assertTrue(Objects.requireNonNull(response.getBody()).isResult());
49      verify(userServiceImpl, times(1)).addPhone(1L, "1234567890");
50    }
51
52    @Test
53    void testAddPayment() {
54      AddPaymentRequest request = new AddPaymentRequest();
55      request.setUserId(1L);
56      request.setCardNum("1234567890123456");
57      request.setCardDate("12/25");
58      request.setCardCVV("123");
59      AddPaymentResponse expectedResponse = new AddPaymentResponse();
60      expectedResponse.setResult(true);
61
62      when(userServiceImpl.addPayment(1L, "1234567890123456", "12/25", "123")).thenReturn(ResponseEntity.ok(expectedResponse));
63
64      ResponseEntity<AddPaymentResponse> response = userController.addPayment(request);
65
66      assertEquals(HttpStatus.OK, response.getStatusCode());
67      assertTrue(Objects.requireNonNull(response.getBody()).isResult());
68      verify(userServiceImpl, times(1)).addPayment(1L, "1234567890123456", "12/25", "123");
69    }
70
71  }
```

108 Business-logic-main-Consumer/demo/src/test/java/com/example/demo/UserServiceTests.java

```
@@ -0,0 +1,108 @@
1   package com.example.demo;
2
3   import com.example.demo.dao.order.OrderEntity;
4   import com.example.demo.dao.order.OrderRepository;
5   import com.example.demo.dao.payment.PaymentsEntity;
```

```
25  @SpringBootTest
26  public class UserServiceTests {
27
28    @Mock
29    private PaymentsRepository paymentRepository;
30
31    @Mock
32    private UserRepository userRepository;
33
34    @Mock
35    private OrderRepository orderRepository;
36
37    @InjectMocks
38    private UserServiceImpl userService;
39
40    @BeforeEach
41    void setUp() {
42      MockitoAnnotations.initMocks(this);
43    }
44
45    @Test
46    void testAddPhoneWithValidIdAndPhoneNumber() {
47      UserEntity user = new UserEntity();
48      user.setId(1L);
49      when(userRepository.findById(1L)).thenReturn(Optional.of(user));
50
51      ResponseEntity<AddPhoneResponse> response = userService.addPhone(1L, "1234567890");
52
53      assertTrue(Objects.requireNonNull(response.getBody()).isResult());
54      verify(userRepository, times(1)).save(user);
55    }
56
57    @Test
58    void testAddPhoneWithInvalidPhoneNumber() {
59      ResponseEntity<AddPhoneResponse> response = userService.addPhone(1L, "invalid_number");
60
61      assertFalse(Objects.requireNonNull(response.getBody()).isResult());
62      verifyNoInteractions(userRepository);
```



```
    @Test
    void testKafkaListenersListenerOrders1WithValidData() {
        UserEntity user = new UserEntity();
        user.setId(1L);
        when(userRepository.findById(1L)).thenReturn(Optional.of(user));

        PaymentsEntity paymentEntity = new PaymentsEntity();
        paymentEntity.setId(1L);
        when(paymentRepository.findByCardNum("1234567890123456")).thenReturn(Optional.of(paymentEntity));

        String data = "{\"userId\":1,\"cardNum\":\"1234567890123456\",\"cardDate\":\"12/25\",\"cardCVV\":\"123\",\"cost\":500.0,\"address\":\"123 Street\"}";

        assertTrue(userService.new KafkaListeners().listenerOrders1(data));
        verify(orderRepository, times(1)).save(any(OrderEntity.class));
    }

    @Test
    void testKafkaListenersListenerOrders1WithInvalidData() {
        assertFalse(userService.new KafkaListeners().listenerOrders1("invalid_data"));
        verifyNoInteractions(orderRepository);
    }

    @Test
    void testKafkaListenersListenerOrders2WithValidData() {
        UserEntity user = new UserEntity();
        user.setId(1L);
        when(userRepository.findById(1L)).thenReturn(Optional.of(user));

        PaymentsEntity paymentEntity = new PaymentsEntity();
        paymentEntity.setId(1L);
        when(paymentRepository.findByCardNum("1234567890123456")).thenReturn(Optional.of(paymentEntity));

        String data = "{\"userId\":1,\"cardNum\":\"1234567890123456\",\"cardDate\":\"12/25\",\"cardCVV\":\"123\",\"cost\":500.0,\"address\":\"123 Street\"}";

        assertTrue(userService.new KafkaListeners().listenerOrders2(data));
        verify(orderRepository, times(1)).save(any(OrderEntity.class));
    }

    @Test
    @Test
    void testKafkaListenersListenerOrders2WithInvalidData() {
        assertFalse(userService.new KafkaListeners().listenerOrders2("invalid_data"));
        verifyNoInteractions(orderRepository);
    }
}
```