

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.

Р34131

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:

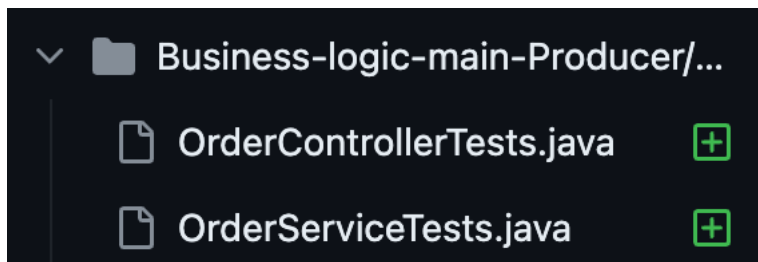
1. Использовали Junit для написания тестов
2. Использовали Mockito для мока необходимых компонентов

Ссылка на коммит:

Для Producer: <https://github.com/lcerzack/blps-ref/commit/e3a3a5c7ffe8b02b279b1110ded209d8da231cbf>

Для Consumer: <https://github.com/lcerzack/blps-ref/commit/2425cbc33b09640d36021b8c6fdbd39d69f9ffeb>

Producer:



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

...
1 package com.example.demo;
2
3 import com.example.demo.controller.OrderController;
4 import com.example.demo.dto.requests.PerformPaymentRequest;
5 import com.example.demo.dto.responses.CheckSmsResponse;
6 import com.example.demo.dto.responses.CheckSumResponse;
7 import com.example.demo.dto.responses.PerformPaymentResponse;
8 import com.example.demo.service.impl.OrderServiceImpl;
9 import java.util.Objects;
10 import org.junit.jupiter.api.BeforeEach;
11 import org.junit.jupiter.api.Test;
12 import org.mockito.InjectMocks;
13 import org.mockito.Mock;
14 import org.mockito.MockitoAnnotations;
15 import org.springframework.boot.test.context.SpringBootTest;
16 import org.springframework.http.HttpStatus;
17 import org.springframework.http.ResponseEntity;
18
19 import static org.junit.jupiter.api.Assertions.*;
20 import static org.mockito.Mockito.*;
21
22 @SpringBootTest
23 class OrderControllerTest {
24
25     @Mock
26     private OrderServiceImpl orderServiceImpl;
27
28     @InjectMocks
29     private OrderController orderController;
30
31     @BeforeEach
32     void setUp() {
33         MockitoAnnotations.initMocks(this);
34     }
35
36     @Test
37     void testCheckSum() {
38         int id = 1;
39         int sum = 120;
40         CheckSumResponse expectedResponse = new CheckSumResponse();
41         expectedResponse.setResult(true);
42
43         when(orderServiceImpl.checkSum(1L, 120)).thenReturn(ResponseEntity.ok(expectedResponse));
44
45         ResponseEntity<CheckSumResponse> response = orderController.checkSum(id, sum);
46
47         assertEquals(HttpStatus.OK, response.getStatusCode());
48         assertTrue(Objects.requireNonNull(response.getBody()).isResult());
49         verify(orderServiceImpl, times(1)).checkSum(1L, 120);
50     }
51
52     @Test
53     void testCheckSms() {
54         int id = 1;
55         String phone = "1234567890";
56         String sms = "1234";
57         CheckSmsResponse expectedResponse = new CheckSmsResponse();
58         expectedResponse.setResult(true);
59
60         when(orderServiceImpl.checkSms(1, "1234567890", "1234")).thenReturn(ResponseEntity.ok(expectedResponse));
61
62         ResponseEntity<CheckSmsResponse> response = orderController.checkSms(id, phone, sms);
63
64         assertEquals(HttpStatus.OK, response.getStatusCode());
65         assertTrue(Objects.requireNonNull(response.getBody()).isResult());
66         verify(orderServiceImpl, times(1)).checkSms(1, "1234567890", "1234");
67     }
68
69     @Test
```

```

69  @Test
70  void testPerformPayment() {
71      PerformPaymentRequest request = new PerformPaymentRequest(1L, "1234567890123456", "12/25", "123", 100.0, "Address");
72      PerformPaymentResponse expectedResponse = new PerformPaymentResponse();
73      expectedResponse.setResult(true);
74
75      when(orderServiceImpl.performPayment(1L, "1234567890123456", "12/25", "123", 100.0, "Address"))
76          .thenReturn(ResponseEntity.ok(expectedResponse));
77
78      ResponseEntity<PerformPaymentResponse> response = orderController.performPayment(request);
79
80      assertEquals(HttpStatus.OK, response.getStatusCode());
81      assertTrue(Objects.requireNonNull(response.getBody()).isResult());
82      verify(orderServiceImpl, times(1)).performPayment(1L, "1234567890123456", "12/25", "123", 100.0, "Address");
83  }
84  }

```

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

```

...  ...  @@ -0,0 +1,104 @@
1  package com.example.demo;
2
3  + import static org.junit.jupiter.api.Assertions.assertFalse;
4  import static org.junit.jupiter.api.Assertions.assertTrue;
5  import static org.mockito.ArgumentMatchers.any;
6  import static org.mockito.ArgumentMatchers.eq;
7  import static org.mockito.Mockito.times;
8  import static org.mockito.Mockito.verify;
9  import static org.mockito.Mockito.verifyNoInteractions;
10
11  import com.example.demo.dto.requests.PerformPaymentRequest;
12  import com.example.demo.dto.responses.CheckSmsResponse;
13  import com.example.demo.dto.responses.CheckSumResponse;
14  import com.example.demo.dto.responses.PerformPaymentResponse;
15  import com.example.demo.service.impl.OrderServiceImpl;
16  import java.util.Objects;
17  import org.junit.jupiter.api.BeforeEach;
18  import org.junit.jupiter.api.Test;
19  import org.mockito.InjectMocks;
20
21  @SpringBootTest
22  public class OrderServiceTests {
23
24      @Mock
25      private KafkaTemplate<String, PerformPaymentRequest> kafkaTemplate;
26
27      @InjectMocks
28      private OrderServiceImpl orderService;
29
30      @BeforeEach
31      void setUp() {
32          MockitoAnnotations.initMocks(this);
33      }
34
35      @Test
36      void testCheckSum() {
37          ResponseEntity<CheckSumResponse> response = orderService.checkSum(123L, 150.0);
38          + assertTrue(Objects.requireNonNull(response.getBody()).isResult());
39      }
40
41      @Test
42      void testCheckSms() {
43          ResponseEntity<CheckSmsResponse> response = orderService.checkSms(123, "1234567890", "1234");
44          assertTrue(Objects.requireNonNull(response.getBody()).isResult());
45      }
46
47      @Test
48      void testPerformPaymentValid() {
49          ResponseEntity<PerformPaymentResponse> response = orderService.performPayment(1L, "1234567890123456",
50              "12/25", "123", 500.0, "123 Street");
51          assertTrue(Objects.requireNonNull(response.getBody()).isResult());
52          verify(kafkaTemplate, times(1)).send(eq("perform-test3"), any(PerformPaymentRequest.class));
53      }
54
55      @Test
56      void testPerformPaymentInvalid() {
57          ResponseEntity<PerformPaymentResponse> response = orderService.performPayment(1L, "123",
58              "12/25", "123", 500.0, "123 Street");
59          assertFalse(Objects.requireNonNull(response.getBody()).isResult());
60      }

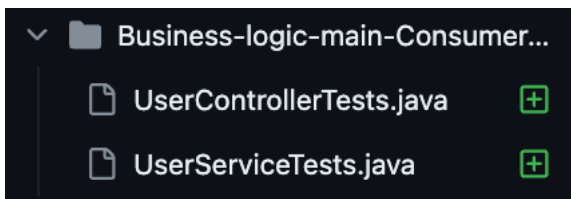
```

```

68     @Test
69     void testPerformPaymentWithInvalidCardCVV() {
70         ResponseEntity<PerformPaymentResponse> response = orderService.performPayment(
71             1L, "1234567890123456", "12/25", "123", 500.0, "123 Street");
72
73         assertFalse(Objects.requireNonNull(response.getBody()).isResult());
74         verifyNoInteractions(kafkaTemplate);
75     }
76
77     @Test
78     void testPerformPaymentWithInvalidCardNum() {
79         ResponseEntity<PerformPaymentResponse> response = orderService.performPayment(
80             1L, "123", "12/25", "123", 500.0, "123 Street");
81
82         assertFalse(Objects.requireNonNull(response.getBody()).isResult());
83         verifyNoInteractions(kafkaTemplate);
84     }
85
86     @Test
87     void testPerformPaymentWithInvalidCost() {
88         ResponseEntity<PerformPaymentResponse> response = orderService.performPayment(
89             1L, "1234567890123456", "12/25", "123", -50.0, "123 Street");
90
91         assertFalse(Objects.requireNonNull(response.getBody()).isResult());
92         verifyNoInteractions(kafkaTemplate);
93     }
94
95     @Test
96     void testPerformPaymentWithValidPaymentDetails() {
97         ResponseEntity<PerformPaymentResponse> response = orderService.performPayment(
98             1L, "1234567890123456", "12/25", "123", 500.0, "123 Street");
99
100         assertTrue(Objects.requireNonNull(response.getBody()).isResult());
101         verify(kafkaTemplate, times(1)).send(eq("perform-test3"), any(PerformPaymentRequest.class));
102     }
103
104 }

```

Consumer:



```

71 Business-logic-main-Consumer/demo/src/test/java/com/example/demo/UserControllerTests.java
4 import com.example.demo.dto.requests.AddPaymentRequest;
5 import com.example.demo.dto.requests.AddPhoneRequest;
6 import com.example.demo.dto.responses.*;
7 import com.example.demo.service.impl.UserServiceImpl;
8 import java.util.Objects;
9 import org.junit.jupiter.api.BeforeEach;
10 import org.junit.jupiter.api.Test;
11 import org.mockito.InjectMocks;
12 import org.mockito.Mock;
13 import org.mockito.MockitoAnnotations;
14 import org.springframework.boot.test.context.SpringBootTest;
15 import org.springframework.http.HttpStatus;
16 import org.springframework.http.ResponseEntity;
17
18 import static org.junit.jupiter.api.Assertions.*;
19 import static org.mockito.Mockito.*;
20
21 @SpringBootTest
22 class UserControllerTests {
23
24     @Mock
25     private UserServiceImpl userServiceImpl;
26
27     @InjectMocks
28     private UserController userController;
29
30     @BeforeEach
31     void setUp() {
32         MockitoAnnotations.openMocks(this);
33     }
34
35     @Test
36     void testAddPhone() {
37         AddPhoneRequest request = new AddPhoneRequest();
38         request.setUserId(1L);
39         request.setPhoneNumber("1234567890");
40         AddPhoneResponse expectedResponse = new AddPhoneResponse();
41         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

```
...
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);
63     }
64 }
```

```

65     @Test
66     void testKafkaListenersListenerOrders1WithValidData() {
67         UserEntity user = new UserEntity();
68         user.setId(1L);
69         when(userRepository.findById(1L)).thenReturn(Optional.of(user));
70
71         PaymentsEntity paymentEntity = new PaymentsEntity();
72         paymentEntity.setId(1L);
73         when(paymentRepository.findByIdCardNum("1234567890123456")).thenReturn(Optional.of(paymentEntity));
74
75         String data = "{\"userId\":\"1\",\"cardNum\":\"1234567890123456\",\"cardDate\":\"12/25\",\"cardCVV\":\"123\",\"cost\":500.0,\"address\":\"123 Street\"}";
76
77         assertTrue(userService.new KafkaListeners().listenerOrders1(data));
78         verify(orderRepository, times(1)).save(any(OrderEntity.class));
79     }
80
81     @Test
82     void testKafkaListenersListenerOrders1WithInvalidData() {
83         assertFalse(userService.new KafkaListeners().listenerOrders1("invalid_data"));
84         verifyNoInteractions(orderRepository);
85     }
86
87     @Test
88     void testKafkaListenersListenerOrders2WithValidData() {
89         UserEntity user = new UserEntity();
90         user.setId(1L);
91         when(userRepository.findById(1L)).thenReturn(Optional.of(user));
92
93         PaymentsEntity paymentEntity = new PaymentsEntity();
94         paymentEntity.setId(1L);
95         when(paymentRepository.findByIdCardNum("1234567890123456")).thenReturn(Optional.of(paymentEntity));
96
97         String data = "{\"userId\":\"1\",\"cardNum\":\"1234567890123456\",\"cardDate\":\"12/25\",\"cardCVV\":\"123\",\"cost\":500.0,\"address\":\"123 Street\"}";
98
99         assertTrue(userService.new KafkaListeners().listenerOrders2(data));
100        verify(orderRepository, times(1)).save(any(OrderEntity.class));
101    }
102
103    @Test
104    void testKafkaListenersListenerOrders2WithInvalidData() {
105        assertFalse(userService.new KafkaListeners().listenerOrders2("invalid_data"));
106        verifyNoInteractions(orderRepository);
107    }
108 }

```