Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

Лабораторная работа №5

по дисциплине «Программирование на языках высокого уровня»

« Batch data processing & Testing»

Выполнил: Проверил:

Снитко Д.А. ассистент Скиба И.Г.

гр.250501

Минск 2024

**1. Постановка задачи**

1.Добавить POST метод для работы со списком параметров (передаются в теле запроса) для bulk операций, организовать работу сервиса используя Java 8 (Stream API, лямбда-выражения).

2. Покрытие Unit-тестами на >80% (бизнес-логика).

**2. Структура проекта**

В проекте используется послойная архитектура из нескольких пакетов,

которые отвечают за определенные функции.

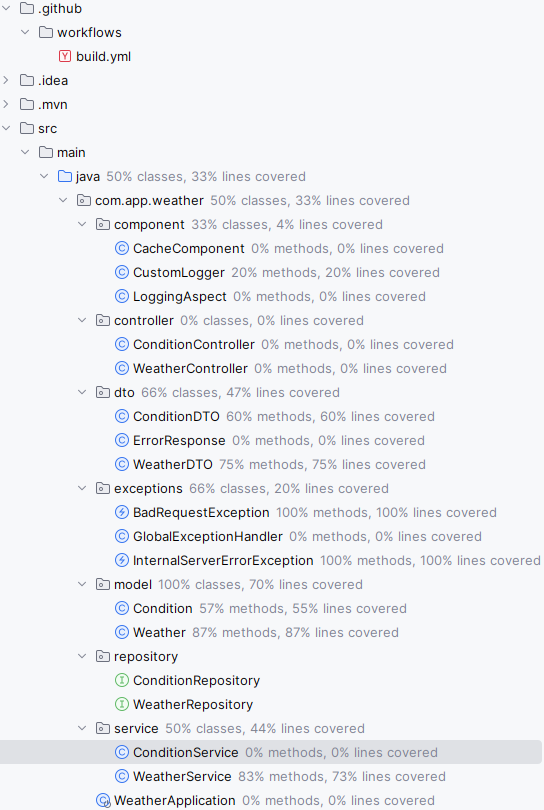


Рисунок 2.1 – Структура проекта

**3. Листинг кода**

Файл ConditionServiceTest.java

class ConditionServiceTest {

@Mock

private ConditionRepository conditionRepository;

@Mock

private CacheComponent cache;

@Mock

private CustomLogger customLogger;

private ConditionService conditionService;

@BeforeEach

void setup() {

MockitoAnnotations.initMocks(this);

conditionService = new ConditionService(conditionRepository, cache, customLogger);

}

@Test

void testCreateCondition() {

Condition condition = new Condition();

condition.setId(1L);

condition.setText("Sunny");

when(conditionRepository.save(condition)).thenReturn(condition);

Condition createdCondition = conditionService.createCondition(condition);

assertEquals(condition, createdCondition);

verify(conditionRepository, times(1)).save(condition);

verify(cache, times(1)).put(condition.getId().toString(), createdCondition);

}

@Test

void testCreateConditionWithExistingTextThrowsBadRequestException() {

Condition condition = new Condition();

condition.setText("Sunny");

when(conditionRepository.existsByText(condition.getText())).thenReturn(true);

assertThrows(BadRequestException.class, () -> conditionService.createCondition(condition));

verify(conditionRepository, never()).save(condition);

verify(cache, never()).put(anyString(), any(Condition.class));

}

@Test

void testUpdateCondition() {

Condition existingCondition = new Condition();

existingCondition.setId(1L);

existingCondition.setText("Sunny");

ConditionDTO conditionDTO = new ConditionDTO();

conditionDTO.setText("Cloudy");

when(conditionRepository.findById(existingCondition.getId())).thenReturn(Optional.of(existingCondition));

when(conditionRepository.save(existingCondition)).thenReturn(existingCondition);

Condition updatedCondition = conditionService.updateCondition(existingCondition.getId(), conditionDTO);

assertEquals(conditionDTO.getText(), updatedCondition.getText());

verify(conditionRepository, times(1)).save(existingCondition);

}

@Test

void testUpdateConditionWithNonExistingIdThrowsBadRequestException() {

ConditionDTO conditionDTO = new ConditionDTO();

conditionDTO.setText("Cloudy");

when(conditionRepository.findById(1L)).thenReturn(Optional.empty());

assertThrows(InternalServerErrorException.class, () -> conditionService.updateCondition(1L, conditionDTO));

verify(conditionRepository, never()).save(any(Condition.class));

verify(cache, never()).put(anyString(), any(Condition.class));

}

@Test

void testDeleteCondition() {

Condition condition = new Condition();

condition.setId(1L);

condition.setText("Test condition");

when(conditionRepository.save(condition)).thenReturn(condition);

when(conditionRepository.findById(condition.getId())).thenReturn(Optional.of(condition));

when(conditionRepository.existsById(condition.getId())).thenReturn(true);

conditionService.createCondition(condition);

Long id = condition.getId();

conditionService.deleteCondition(id);

verify(conditionRepository, times(1)).deleteById(id);

verify(cache, times(1)).remove(id.toString());

}

@Test

void testGetConditionById() {

Condition condition = new Condition();

condition.setId(1L);

condition.setText("Sunny");

when(conditionRepository.findById(condition.getId())).thenReturn(Optional.of(condition));

Condition foundCondition = conditionService.getConditionById(condition.getId());

assertEquals(condition, foundCondition);

verify(conditionRepository, times(1)).findById(condition.getId());

verify(cache, times(1)).put(condition.getId().toString(), foundCondition);

}

@Test

void testGetConditionByIdWithNonExistingIdThrowsBadRequestException() {

Long id = 1L;

when(conditionRepository.findById(id)).thenReturn(Optional.empty());

assertThrows(InternalServerErrorException.class, () -> conditionService.getConditionById(id));

verify(cache, never()).put(anyString(), any(Condition.class));

}

@Test

void testGetAllConditions() {

List<Condition> conditions = new ArrayList<>();

conditions.add(new Condition());

conditions.add(new Condition());

when(conditionRepository.findAll()).thenReturn(conditions);

assertThrows(InternalServerErrorException.class, () -> conditionService.getAllConditions());

verify(conditionRepository, times(1)).findAll();

}

@Test

void testGetConditionByText() {

String text = "Sunny";

Condition condition = new Condition();

condition.setId(1L); // устанавливаем идентификатор

condition.setText(text);

when(conditionRepository.findByText(text)).thenReturn(condition);

Condition foundCondition = conditionService.getConditionByText(text);

assertEquals(condition, foundCondition);

verify(conditionRepository, times(1)).findByText(text);

verify(cache, times(1)).put(foundCondition.getId().toString(), foundCondition);

}

@Test

void testGetConditionByTextWithNonExistingTextReturnsNull() {

String text = "Sunny";

when(conditionRepository.findByText(text)).thenReturn(null);

Condition condition = conditionService.getConditionByText(text);

assertNull(condition);

verify(conditionRepository, times(1)).findByText(text);

verify(cache, never()).put(anyString(), any(Condition.class));

}

@Test

void testCreateConditionBulk() {

List<ConditionDTO> conditionDTOs = new ArrayList<>();

conditionDTOs.add(new ConditionDTO(null, "Sunny"));

conditionDTOs.add(new ConditionDTO(null, "Cloudy"));

conditionDTOs.add(new ConditionDTO(null, "Rainy"));

when(conditionRepository.existsByText(anyString())).thenReturn(false);

when(conditionRepository.save(any(Condition.class))).thenAnswer(invocation -> invocation.getArgument(0));

List<Condition> createdConditions = conditionService.createConditionBulk(conditionDTOs);

assertEquals(3, createdConditions.size());

verify(conditionRepository, times(3)).save(any(Condition.class));

verify(cache, never()).put(anyString(), any(Condition.class));

}

@Test

void testUpdateNonExistingConditionThrowsBadRequestException() {

ConditionDTO conditionDTO = new ConditionDTO(null, "Cloudy");

when(conditionRepository.findById(1L)).thenReturn(Optional.empty());

assertThrows(InternalServerErrorException.class, () -> conditionService.updateCondition(1L, conditionDTO));

verify(conditionRepository, never()).save(any(Condition.class));

verify(cache, never()).put(anyString(), any(Condition.class));

}

@Test

void testUpdateConditionWithExistingValueThrowsBadRequestException() {

Condition existingCondition = new Condition();

existingCondition.setId(1L);

existingCondition.setText("Sunny");

ConditionDTO conditionDTO = new ConditionDTO(null, "Cloudy");

when(conditionRepository.findById(existingCondition.getId())).thenReturn(Optional.of(existingCondition));

when(conditionRepository.existsByTextAndIdNot("Cloudy", existingCondition.getId())).thenReturn(true);

assertThrows(BadRequestException.class, () -> conditionService.updateCondition(1L, conditionDTO));

verify(conditionRepository, never()).save(any(Condition.class));

}

@Test

void testDeleteNonExistingConditionThrowsBadRequestException() {

when(conditionRepository.existsById(1L)).thenReturn(false);

assertThrows(BadRequestException.class, () -> conditionService.deleteCondition(1L));

verify(conditionRepository, never()).deleteById(anyLong());

verify(cache, never()).remove(anyString());

}

@Test

void testGetConditionByTextWithNonExistingTextThrowsInternalServerErrorException() {

String text = "Sunny";

when(conditionRepository.findByText(text)).thenThrow(new RuntimeException());

assertThrows(InternalServerErrorException.class, () -> conditionService.getConditionByText(text));

verify(conditionRepository, times(1)).findByText(text);

verify(cache, never()).put(anyString(), any(Condition.class));

}

}

Файл WeatherServiceTest.java

class WeatherServiceTest {

@Mock

private WeatherRepository weatherRepository;

@Mock

private ConditionRepository conditionRepository;

@Mock

private ConditionService conditionService;

@Mock

private CacheComponent cacheComponent;

@Mock

private CustomLogger customLogger;

@Captor

private ArgumentCaptor<Weather> weatherArgumentCaptor;

private WeatherService weatherService;

@BeforeEach

void setUp() {

MockitoAnnotations.initMocks(this);

weatherService = new WeatherService(weatherRepository, conditionService, cacheComponent, customLogger);

}

@Test

void testCreateWeatherBulkSuccess() {

List<WeatherDTO> weatherDTOs = new ArrayList<>();

WeatherDTO weatherDTO1 = new WeatherDTO();

weatherDTO1.setCity("London");

weatherDTO1.setTemperature(20.0);

weatherDTO1.setCondition(new ConditionDTO());

weatherDTO1.getCondition().setText("Cloudy");

weatherDTOs.add(weatherDTO1);

WeatherDTO weatherDTO2 = new WeatherDTO();

weatherDTO2.setCity("Paris");

weatherDTO2.setTemperature(25.0);

weatherDTO2.setCondition(new ConditionDTO());

weatherDTO2.getCondition().setText("Sunny");

weatherDTOs.add(weatherDTO2);

Condition condition1 = new Condition();

condition1.setText("Cloudy");

Condition condition2 = new Condition();

condition2.setText("Sunny");

when(conditionService.getConditionByText("Cloudy")).thenReturn(condition1);

when(conditionService.getConditionByText("Sunny")).thenReturn(condition2);

when(weatherRepository.save(any(Weather.class))).thenAnswer(invocation -> invocation.getArgument(0));

List<Weather> createdWeathers = weatherService.createWeatherBulk(weatherDTOs);

assertEquals(2, createdWeathers.size());

verify(conditionService, times(2)).getConditionByText(anyString());

verify(weatherRepository, times(2)).save(any(Weather.class));

}

@Test

void testCreateWeatherBulkCityAlreadyExists() {

List<WeatherDTO> weatherDTOs = new ArrayList<>();

WeatherDTO weatherDTO1 = new WeatherDTO();

weatherDTO1.setCity("London");

weatherDTO1.setTemperature(20.0);

weatherDTO1.setCondition(new ConditionDTO());

weatherDTO1.getCondition().setText("Cloudy");

weatherDTOs.add(weatherDTO1);

WeatherDTO weatherDTO2 = new WeatherDTO();

weatherDTO2.setCity("London");

weatherDTO2.setTemperature(25.0);

weatherDTO2.setCondition(new ConditionDTO());

weatherDTO2.getCondition().setText("Sunny");

weatherDTOs.add(weatherDTO2);

when(weatherRepository.existsByCity("London")).thenReturn(true);

assertThrows(BadRequestException.class, () -> weatherService.createWeatherBulk(weatherDTOs));

verify(weatherRepository, times(1)).existsByCity("London");

verify(conditionService, never()).getConditionByText(anyString());

verify(weatherRepository, never()).save(any(Weather.class));

}

@Test

void testCreateWeatherBulkTransaction() {

List<WeatherDTO> weatherDTOs = new ArrayList<>();

WeatherDTO weatherDTO1 = new WeatherDTO();

weatherDTO1.setCity("London");

weatherDTO1.setTemperature(20.0);

weatherDTO1.setCondition(new ConditionDTO());

weatherDTO1.getCondition().setText("Cloudy");

weatherDTOs.add(weatherDTO1);

WeatherDTO weatherDTO2 = new WeatherDTO();

weatherDTO2.setCity("Paris");

weatherDTO2.setTemperature(25.0);

weatherDTO2.setCondition(new ConditionDTO());

weatherDTO2.getCondition().setText("Sunny");

weatherDTOs.add(weatherDTO2);

Condition condition1 = new Condition();

condition1.setText("Cloudy");

when(conditionService.getConditionByText("Cloudy")).thenReturn(condition1);

when(conditionService.getConditionByText("Sunny")).thenThrow(new RuntimeException());

when(weatherRepository.save(any(Weather.class))).thenAnswer(invocation -> invocation.getArgument(0));

assertThrows(RuntimeException.class, () -> weatherService.createWeatherBulk(weatherDTOs));

verify(conditionService, times(2)).getConditionByText(anyString());

verify(weatherRepository, times(1)).save(any(Weather.class));

}

@Test

void testCreateWeatherWithConditionWhenConditionExistsShouldCreateWeather() throws BadRequestException, InternalServerErrorException {

// Подготовка данных

WeatherDTO weatherDTO = new WeatherDTO();

weatherDTO.setCity("City");

weatherDTO.setCondition(new ConditionDTO());

weatherDTO.getCondition().setText("Condition");

Condition condition = new Condition();

condition.setText("Condition");

Weather weather = new Weather();

weather.setCity("City");

weather.setCondition(condition);

when(conditionService.getConditionByText(weatherDTO.getCondition().getText())).thenReturn(condition);

when(weatherRepository.save(any(Weather.class))).thenReturn(weather);

// Вызов метода

Weather result = weatherService.createWeatherWithCondition(weatherDTO);

// Проверка результата

assertEquals(weather, result);

verify(conditionService, times(1)).getConditionByText(weatherDTO.getCondition().getText());

verify(weatherRepository, times(1)).save(weatherArgumentCaptor.capture());

Weather capturedWeather = weatherArgumentCaptor.getValue();

assertEquals("City", capturedWeather.getCity());

assertEquals(condition, capturedWeather.getCondition());

}

@Test

void testCreateWeatherWithConditionWhenConditionDoesNotExistShouldThrowBadRequestException() {

// Подготовка данных

WeatherDTO weatherDTO = new WeatherDTO();

weatherDTO.setCity("City");

weatherDTO.setCondition(new ConditionDTO());

weatherDTO.getCondition().setText("Condition");

when(conditionService.getConditionByText("NonExistentCondition")).thenThrow(new BadRequestException("Condition not found"));

// Вызов метода и проверка исключения

assertThrows(InternalServerErrorException.class, () -> weatherService.createWeatherWithCondition(weatherDTO));

verify(conditionService, times(1)).getConditionByText(weatherDTO.getCondition().getText());

verify(weatherRepository, never()).save(any(Weather.class));

}

@Test

void testCreateWeatherWithConditionWhenWeatherForCityAlreadyExistsShouldThrowBadRequestException() {

// Подготовка данных

WeatherDTO weatherDTO = new WeatherDTO();

weatherDTO.setCity("City");

weatherDTO.setCondition(new ConditionDTO());

weatherDTO.getCondition().setText("Condition");

Weather existingWeather = new Weather();

existingWeather.setCity("City");

existingWeather.setCondition(new Condition());

existingWeather.getCondition().setText("Condition");

when(weatherRepository.findByCity(weatherDTO.getCity())).thenReturn(existingWeather);

// Вызов метода и проверка исключения

assertThrows(InternalServerErrorException.class, () -> weatherService.createWeatherWithCondition(weatherDTO));

verify(weatherRepository, times(1)).findByCity(weatherDTO.getCity());

verify(weatherRepository, never()).save(any(Weather.class));

}

@Test

void testUpdateWeatherWhenWeatherExistsShouldUpdateWeather() throws BadRequestException, InternalServerErrorException {

// Подготовка данных

WeatherDTO weatherDTO = new WeatherDTO();

weatherDTO.setCity("City");

weatherDTO.setTemperature(25.0);

weatherDTO.setCondition(new ConditionDTO());

weatherDTO.getCondition().setText("Condition");

Condition condition = new Condition();

condition.setText("Condition");

Weather existingWeather = new Weather();

existingWeather.setId(1L);

existingWeather.setCity("City");

existingWeather.setTemperature(20.0);

existingWeather.setCondition(condition);

Weather updatedWeather = new Weather();

updatedWeather.setId(1L);

updatedWeather.setCity("City");

updatedWeather.setTemperature(25.0);

updatedWeather.setCondition(condition);

when(weatherRepository.findById(1L)).thenReturn(Optional.of(existingWeather));

when(conditionService.getConditionByText(weatherDTO.getCondition().getText())).thenReturn(condition);

when(weatherRepository.save(any(Weather.class))).thenReturn(updatedWeather);

// Вызов метода

Weather result = weatherService.updateWeather(1L, weatherDTO);

// Проверка результата

assertEquals(updatedWeather, result);

verify(weatherRepository, times(1)).findById(1L);

verify(conditionService, times(1)).getConditionByText(weatherDTO.getCondition().getText());

verify(weatherRepository, times(1)).save(weatherArgumentCaptor.capture());

Weather capturedWeather = weatherArgumentCaptor.getValue();

assertEquals("City", capturedWeather.getCity());

assertEquals(25.0, capturedWeather.getTemperature());

assertEquals(condition, capturedWeather.getCondition());

}

@Test

void testUpdateWeatherWhenWeatherDoesNotExistShouldThrowBadRequestException() {

// Подготовка данных

WeatherDTO weatherDTO = new WeatherDTO();

weatherDTO.setCity("City");

weatherDTO.setTemperature(25.0);

weatherDTO.setCondition(new ConditionDTO());

weatherDTO.getCondition().setText("Condition");

when(weatherRepository.findById(1L)).thenReturn(Optional.empty());

// Вызов метода и проверка исключения

assertThrows(InternalServerErrorException.class, () -> weatherService.updateWeather(1L, weatherDTO));

verify(weatherRepository, times(1)).findById(1L);

verify(conditionService, never()).getConditionByText(anyString());

verify(weatherRepository, never()).save(any(Weather.class));

}

@Test

void testDeleteWeatherWhenWeatherExistsShouldDeleteWeather() throws BadRequestException, InternalServerErrorException {

// Подготовка данных

Weather existingWeather = new Weather();

existingWeather.setId(1L);

existingWeather.setCity("City");

when(weatherRepository.findById(1L)).thenReturn(Optional.of(existingWeather));

// Вызов метода

weatherService.deleteWeather(1L);

// Проверка результата

verify(weatherRepository, times(1)).findById(1L);

verify(weatherRepository, times(1)).delete(any(Weather.class));

}

@Test

void testDeleteWeatherWhenWeatherDoesNotExistShouldThrowBadRequestException() {

// Подготовка данных

when(weatherRepository.findById(1L)).thenReturn(Optional.empty());

// Вызов метода и проверка исключения

assertThrows(InternalServerErrorException.class, () -> weatherService.deleteWeather(1L));

verify(weatherRepository, times(1)).findById(1L);

verify(weatherRepository, never()).deleteById(anyLong());

}

@Test

void testGetWeatherByIdWhenWeatherExistsShouldReturnWeather() throws BadRequestException, InternalServerErrorException {

// Подготовка данных

Weather existingWeather = new Weather();

existingWeather.setId(1L);

existingWeather.setCity("City");

when(weatherRepository.findById(1L)).thenReturn(Optional.of(existingWeather));

// Вызов метода

Weather result = weatherService.getWeatherById(1L);

// Проверка результата

assertEquals(existingWeather, result);

verify(weatherRepository, times(1)).findById(1L);

}

@Test

void testGetWeatherByIdWhenWeatherDoesNotExistShouldThrowBadRequestException() {

// Подготовка данных

when(weatherRepository.findById(1L)).thenReturn(Optional.empty());

// Вызов метода и проверка исключения

assertThrows(InternalServerErrorException.class, () -> weatherService.getWeatherById(1L));

verify(weatherRepository, times(1)).findById(1L);

}

@Test

void testGetAllWeathersShouldReturnAllWeathers() throws InternalServerErrorException {

// Подготовка данных

List<Weather> weathers = new ArrayList<>();

weathers.add(new Weather());

weathers.add(new Weather());

when(weatherRepository.findAll()).thenReturn(weathers);

// Вызов метода

List<Weather> result = weatherService.getAllWeathers();

// Проверка результата

assertEquals(2, result.size());

verify(weatherRepository, times(1)).findAll();

}

@Test

void testGetAllWeathersWhenWeatherRepositoryThrowsExceptionShouldThrowInternalServerErrorException() {

// Подготовка данных

when(weatherRepository.findAll()).thenThrow(new RuntimeException());

// Вызов метода и проверка исключения

assertThrows(InternalServerErrorException.class, () -> weatherService.getAllWeathers());

verify(weatherRepository, times(1)).findAll();

}

@Test

void testFindByTemperatureWhenWeathersExistsShouldReturnWeathers() throws InternalServerErrorException {

// Подготовка данных

List<Weather> weathers = new ArrayList<>();

weathers.add(new Weather());

weathers.add(new Weather());

when(weatherRepository.findByTemperature(25.0)).thenReturn(weathers);

// Вызов метода

List<WeatherDTO> result = weatherService.findByTemperature(25.0);

// Проверка результата

assertEquals(2, result.size());

verify(weatherRepository, times(1)).findByTemperature(25.0);

}

@Test

void testFindByTemperatureWhenWeatherRepositoryThrowsExceptionShouldThrowInternalServerErrorException() {

// Подготовка данных

when(weatherRepository.findByTemperature(25.0)).thenThrow(new RuntimeException());

// Вызов метода и проверка исключения

assertThrows(InternalServerErrorException.class, () -> weatherService.findByTemperature(25.0));

verify(weatherRepository, times(1)).findByTemperature(25.0);

}

}

**4. Результат программы**

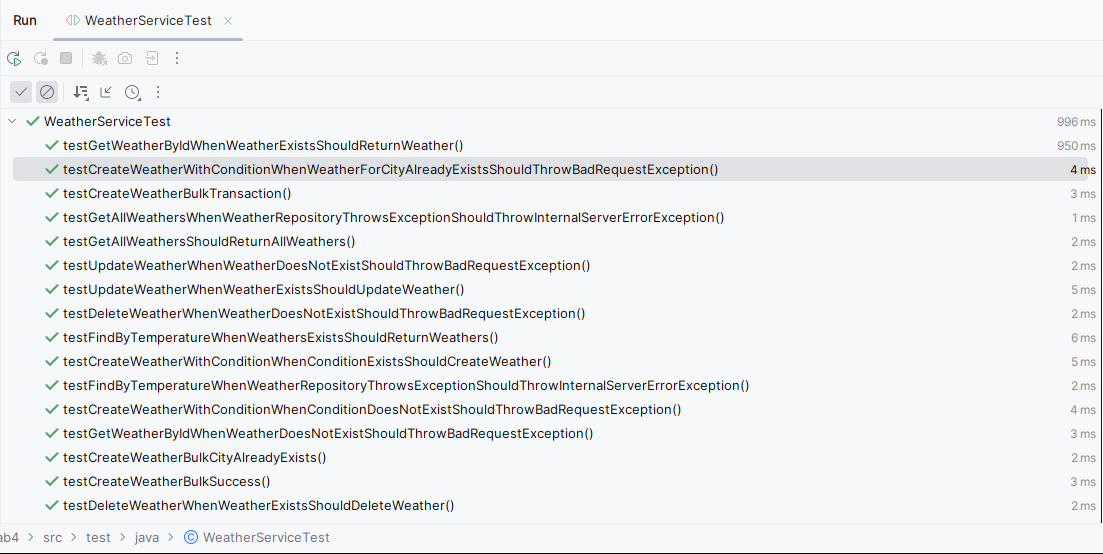


Рисунок 1.1 – успешное выполнение тестов

**5. Заключение**

В результате работы были добавлены POST методы для работы со списком параметров (передающимися в теле запроса) для bulk операций, организована работа сервиса используя Java 8 (Stream API, лямбда-выражения). Было выполнено покрытие Unit-тестами на >80% (бизнес-логика).