|  |  |
| --- | --- |
|  | Министерство образования и науки Российской Федерации |
| Федеральное государственное автономное образовательное учреждение высшего образования |
| «Уральский федеральный университет имени первого Президента России Б. Н. Ельцина» |
| Институт радиоэлектроники и информационных технологий – РтФ |
|  |
|  | |

**ОТЧЕТ**

О ВЫПОЛНЕНИИ

ЛАБОРАТОНОЙ РАБОТЫ № 5

Руководитель ст. пр. Н. А. Архипов

Студент гр. РИМ-230971 Е.В. Треглазов

Екатеринбург 2024

**Цель работы:** доработать простой REST сервис и добавить unit тесты.

1. **Доработка простого REST сервиса**

Расширение класса Request:

package ru.example.MySecondTestAppSpringBoot.model;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

import javax.validation.constraints.\*;

@Data

@Builder

@NoArgsConstructor

@AllArgsConstructor

public class Request {

    @NotBlank

    @Size(max = 32)

    private String uid;

    @NotBlank

    @Size(max = 32)

    private String operationUid;

    private Systems systemName;

    @NotBlank

    private String systemTime;

    private String source;

    private Positions position;

    private Double salary;

    private Double bonus;

    private Integer workDays;

    @Min(1)

    @Max(100000)

    private int communicationId;

    private int templateId;

    private int productCode;

    private int smsCode;

    @Override

    public String toString() {

        return "{" +

                "uid='" + uid + '\'' +

                ", opertionUid='" + operationUid + '\'' +

                ", systemName='" + systemName + '\'' +

                ", systemTime='" + systemTime + '\'' +

                ", source='" + source + '\'' +

                ", communicationId=" + communicationId +

                ", templateId=" + templateId +

                ", productCode=" + productCode +

                ", smsCode=" + smsCode +

                '}';

    }

}

Создание перечисления Positions:

package ru.example.MySecondTestAppSpringBoot.model;

import lombok.Getter;

@Getter

public enum Positions {

    DEV(2.2),

    HR(1.2),

    TL(2.6);

    private final double positionCoefficient;

    Positions(double positionCoefficient) {

        this.positionCoefficient = positionCoefficient;

    }

}

Доработка класса Response:

package ru.example.MySecondTestAppSpringBoot.model;

import lombok.Builder;

import lombok.Data;

@Data

@Builder

public class Response {

    private String uid;

    private String operationUid;

    private String systemTime;

//    private String code;

    private Codes code;

    private Double annualBonus;

//    private String errorCode;

    private ErrorCodes errorCode;

//    private String errorMessage;

    private ErrorMessages errorMessage;

}

Создание сервиса расчета годовой премии AnnualBonusService и класс его реализующий:

package ru.example.MySecondTestAppSpringBoot.service;

import org.springframework.stereotype.Service;

import ru.example.MySecondTestAppSpringBoot.model.Positions;

@Service

public interface AnnualBonusService {

    double calculate(Positions positions, double salary, double bonus, int workDays);

}

package ru.example.MySecondTestAppSpringBoot.service;

import org.springframework.stereotype.Service;

import ru.example.MySecondTestAppSpringBoot.model.Positions;

@Service

public class AnnualBonusServiceImpl implements AnnualBonusService {

    @Override

    public double calculate(Positions positions, double salary, double bonus, int workDays) {

        return salary \* bonus \* 365 \* positions.getPositionCoefficient() / workDays;

    }

}

Доработка класса MyController:

public class MyController {

    private final ValidationService validationService;

    private final ModifyResponseService modifyResponseService;

    private final ModifyRequestService modifyRequestService;

    private final UnsupportedCodeValidation unsupportedCodeValidation;

    private final AnnualBonusServiceImpl annualBonusService;

    @Autowired

    public MyController(ValidationService validationService, UnsupportedCodeValidation unsupportedCodeValidation,

                        @Qualifier("ModifySystemTimeResponseService") ModifyResponseService modifyResponseService,

                        ModifyRequestService modifyRequestService, AnnualBonusServiceImpl annualBonusService) {

        this.validationService = validationService;

        this.unsupportedCodeValidation = unsupportedCodeValidation;

        this.modifyResponseService = modifyResponseService;

        this.modifyRequestService = modifyRequestService;

        this.annualBonusService = annualBonusService;

    }

    @PostMapping(value = "/feedback")

    public ResponseEntity<Response> feedback(@Valid @RequestBody Request request,

                                             BindingResult bindingResult) {

        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'");

        log.info("request: {}", request);

        Response response = Response.builder()

                .uid(request.getUid())

                .operationUid(request.getOperationUid())

                .systemTime(DateTimeUtil.getCustomFormat().format(new Date()))

//                .systemTime(simpleDateFormat.format(new Date()))

                .code(Codes.SUCCESS)

                .errorCode(ErrorCodes.EMPTY)

                .errorMessage(ErrorMessages.EMPTY)

                .build();

        try {

            validationService.isValid(bindingResult);

        } catch (ValidationFailedException e) {

            log.error("error: {}", e.getMessage());

            response.setCode(Codes.FAILED);

//            response.setCode("failed");

            response.setErrorCode(ErrorCodes.VALIDATION\_EXCEPTION);

//            response.setErrorCode("ValidationException");

            response.setErrorMessage(ErrorMessages.VALIDATION);

//            response.setErrorMessage("Ошибка валидации");

            return new ResponseEntity<>(response, HttpStatus.BAD\_REQUEST);

        } catch (Exception e) {

            log.error("error: {}", e.getMessage());

            response.setCode(Codes.FAILED);

//            response.setCode("failed");

            response.setErrorCode(ErrorCodes.UNKNOWN\_EXCEPTION);

//            response.setErrorCode("UnknownException");

            response.setErrorMessage(ErrorMessages.UNKNOWN);

//            response.setErrorMessage("Произошла непредвиденная ошибка");

            return new ResponseEntity<>(response, HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

        try {

            unsupportedCodeValidation.Valid(Integer.parseInt(request.getUid()));

        } catch (UnsupportedCodeException e) {

            log.error("error: {}", e.getMessage());

            response.setCode(Codes.FAILED);

//            response.setCode("failed");

            response.setErrorCode(ErrorCodes.UNSUPPORTED\_EXCEPTION);

//            response.setErrorCode("UnsupportedCodeException");

            response.setErrorMessage(ErrorMessages.UNSUPPORTED);

//            response.setErrorMessage("uid = 123");

            return new ResponseEntity<>(response, HttpStatus.BAD\_REQUEST);

        }

        modifyResponseService.modify(response);

        modifyRequestService.modify(request);

        Double annualBonus = annualBonusService.calculate(request.getPosition(),

                request.getSalary(),

                request.getBonus(),

                request.getWorkDays());

        log.info("response: {}", response);

        return new ResponseEntity<>(modifyResponseService.modify(response), HttpStatus.OK);

    }

}

1. **Создание unit тестов**

Тест метода calculate:

package ru.example.MySecondTestAppSpringBoot.service;

import org.junit.jupiter.api.Test;

import ru.example.MySecondTestAppSpringBoot.model.Positions;

import static org.assertj.core.api.AssertionsForClassTypes.assertThat;

class AnnualBonusServiceImplTest {

    @Test

    void calculate() {

        Positions position = Positions.HR;

        double bonus = 2.0;

        int workDays = 243;

        double salary = 100000.00;

        double result = new AnnualBonusServiceImpl().calculate(position, salary, bonus, workDays);

        double expected = 360493.8271604938;

        assertThat(result).isEqualTo(expected);

    }

}

1. **Реализация дополнительного функционала**

1. Доработайте метод calculate, чтобы в нем вычислялось количество дней в году, в зависимости от того високосный год или нет.

package ru.example.MySecondTestAppSpringBoot.service;

import org.springframework.stereotype.Service;

import ru.example.MySecondTestAppSpringBoot.model.Positions;

import java.util.Calendar;

import java.time.Year;

@Service

public class AnnualBonusServiceImpl implements AnnualBonusService {

    private Integer year = Year.now().getValue();

    public static int DaysInYear(int year) {

        Calendar calendar = Calendar.getInstance();

        calendar.set(Calendar.YEAR, year);

        return calendar.getActualMaximum(Calendar.DAY\_OF\_YEAR);

    }

    @Override

    public double calculate(Positions positions, double salary, double bonus, int workDays) {

        return salary \* bonus \* DaysInYear(this.year) \* positions.getPositionCoefficient() / workDays;

    }

}

2. Расширьте enum Positions, добавьте три дополнительных позиции, а также добавьте поле isManager с типом данных boolean, для менеджеров установите значение true, в прочих случаях false.

package ru.example.MySecondTestAppSpringBoot.model;

import lombok.Getter;

@Getter

public enum Positions {

    DEV(2.2, false),

    HR(1.2, true),

    TL(2.6, true),

    DA(2.3, true),

    DE(1.9, false),

    DS(1.8, false);

    private final double positionCoefficient;

    private final boolean isManager;

    Positions(double positionCoefficient, boolean isManager) {

        this.positionCoefficient = positionCoefficient;

        this.isManager = isManager;

    }

}

3. Напишите метод, вычисляющий квартальную премию, метод работает только для менеджеров и иных управленцев (те, у кого isManager == true), добавьте проверку в метод, на признак управленец это или нет.

package ru.example.MySecondTestAppSpringBoot.service;

import org.springframework.stereotype.Service;

import ru.example.MySecondTestAppSpringBoot.model.Positions;

import java.util.Calendar;

import java.time.Year;

@Service

public class AnnualBonusServiceImpl implements AnnualBonusService {

    private Integer year = Year.now().getValue();

    public static int DaysInYear(int year) {

        Calendar calendar = Calendar.getInstance();

        calendar.set(Calendar.YEAR, year);

        return calendar.getActualMaximum(Calendar.DAY\_OF\_YEAR);

    }

    @Override

    public double calculate(Positions positions, double salary, double bonus, int workDays) {

        return salary \* bonus \* DaysInYear(this.year) \* positions.getPositionCoefficient() / workDays;

    }

    @Override

    public double quarterlyBonus(Positions positions, double salary, double bonus, int workDays) {

        double result = 0.00;

        if (positions.isManager()) {

            result = salary \* bonus \* DaysInYear(this.year) \* positions.getPositionCoefficient() / workDays / 4;

        }

        return result;

    }

}

4. Разработайте unit тест для метода расчета квартальной премии.

   @Test

    void quarterlyBonus() {

        Positions position = Positions.HR;

        double bonus = 2.0;

        int workDays = 243;

        double salary = 100000.00;

        double result = new AnnualBonusServiceImpl().quarterlyBonus(position, salary, bonus, workDays);

        double expected = 90123.4567901234;

        assertThat(result).isEqualTo(expected);

5. Произведите рефакторинг приложения – добавьте комментарии к каждому полю в классах Request и Response в соответствии с описанием из лабораторной работы №2, пример комментария к полю показан на рисунке 1 текущей работы.

Класс Request:

package ru.example.MySecondTestAppSpringBoot.model;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

import javax.validation.constraints.\*;

@Data

@Builder

@NoArgsConstructor

@AllArgsConstructor

public class Request {

    @NotBlank

    @Size(max = 32)

    private String uid;

    @NotBlank

    @Size(max = 32)

    private String operationUid;

    private Systems systemName;

    @NotBlank

    private String systemTime;

    private String source;

    private Positions position;

    private Double salary;

    private Double bonus;

    private Integer workDays;

    @Min(1)

    @Max(100000)

    private int communicationId;

    private int templateId;

    private int productCode;

    private int smsCode;

    @Override

    public String toString() {

        return "{" +

                "uid='" + uid + '\'' +

                ", opertionUid='" + operationUid + '\'' +

                ", systemName='" + systemName + '\'' +

                ", systemTime='" + systemTime + '\'' +

                ", source='" + source + '\'' +

                ", communicationId=" + communicationId +

                ", templateId=" + templateId +

                ", productCode=" + productCode +

                ", smsCode=" + smsCode +

                '}';

    }

}

Класс Response:

package ru.example.MySecondTestAppSpringBoot.model;

import lombok.Builder;

import lombok.Data;

@Data

@Builder

public class Response {

    //Уникальный идентификатор сообщения

    private String uid;

    //Уникальный идентификатор операции

    private String operationUid;

    //Время создания сообщения

    private String systemTime;

    //Код

//    private String code;

    private Codes code;

    //Премия

    private Double annualBonus;

    //Код ошибки

//    private String errorCode;

    private ErrorCodes errorCode;

    //Сообщение об ошибке

//    private String errorMessage;

    private ErrorMessages errorMessage;

}

}

1. **Ответы на вопросы**

2 вариант

2. Что такое smoke тестирование?

Smoke тестирование — это первичное тестирование, которое выполняется для проверки основных функций, оно включает в себя выполнение минимального набора тестов для выявления явных дефектов критичной функциональности.

4. Какие библиотеки используются в Spring Boot для unit тестирования?

Spring Boot использует библиотеку JUnit 5 для unit тестирования. Кроме того, Spring Boot также предоставляет поддержку для других библиотек, таких как Mockito, AssertJ и Hamcrest.

6. Какие аннотациия наиболее распространены при использовании библиотеки JUnit 5?

- @Test: используется для пометки метода как тестового метода.

- @BeforeEach: используется для пометки метода, который должен быть выполнен перед каждым тестовым методом.

- @AfterEach: используется для пометки метода, который должен быть выполнен после каждого тестового метода.

- @BeforeAll: используется для пометки метода, который должен быть выполнен перед запуском всех тестовых методов в классе.

- @AfterAll: используется для пометки метода, который должен быть выполнен после запуска всех тестовых методов в классе.

- @Disabled: используется для отключения выполнения тестового метода или класса.

Репозиторий: https://github.com/kuk86kuk/java\_ufru\_2024.