|  |  |
| --- | --- |
|  | 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 № 4

Руководитель ст. пр. Н. А. Архипов

Студент гр. РИМ-230971 Е.В. Треглазов

Екатеринбург 2024

**Цель работы:** доработать простой REST сервис.

1. **Доработка простого REST сервиса**

Интерфейс ModifyRequestService в сервисе 1 и его реализация:

import org.springframework.stereotype.Service;  
import ru.example.MySecondTestAppSpringBoot.model.Request;  
  
@Service  
public interface ModifyRequestService {  
 void modify(Request request);  
}

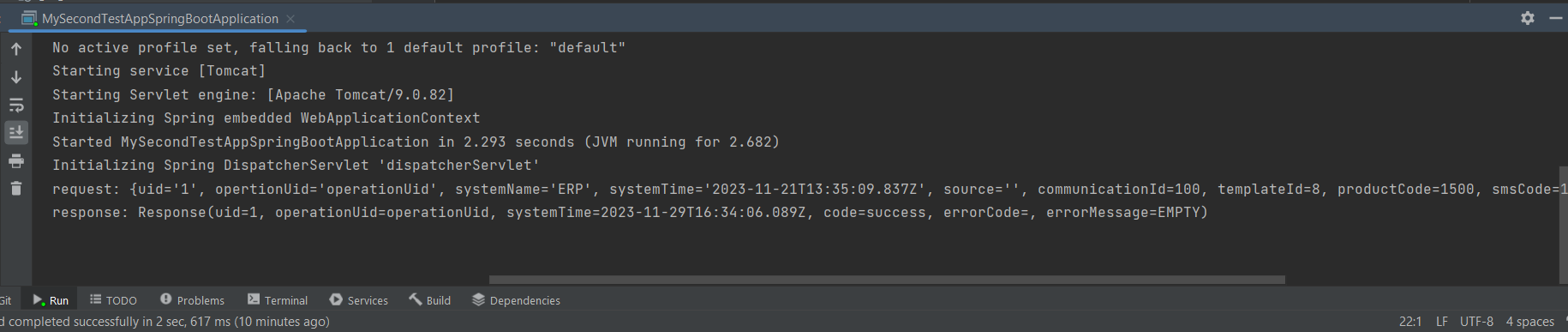
import org.springframework.beans.factory.annotation.Qualifier;  
import org.springframework.stereotype.Service;  
import ru.example.MySecondTestAppSpringBoot.model.Response;  
import ru.example.MySecondTestAppSpringBoot.util.DateTimeUtil;  
  
import java.util.Date;  
  
@Service  
@Qualifier("ModifySystemTimeResponseService")  
public class ModifySystemTimeResponseService implements ModifyResponseService {  
  
 @Override  
 public Response modify(Response response) {  
  
 response.setSystemTime(DateTimeUtil.*getCustomFormat*().format(new Date()));  
  
 return response;  
 }  
}

Внедрение зависимости на сервис ModifyRequestService в класс MyController:

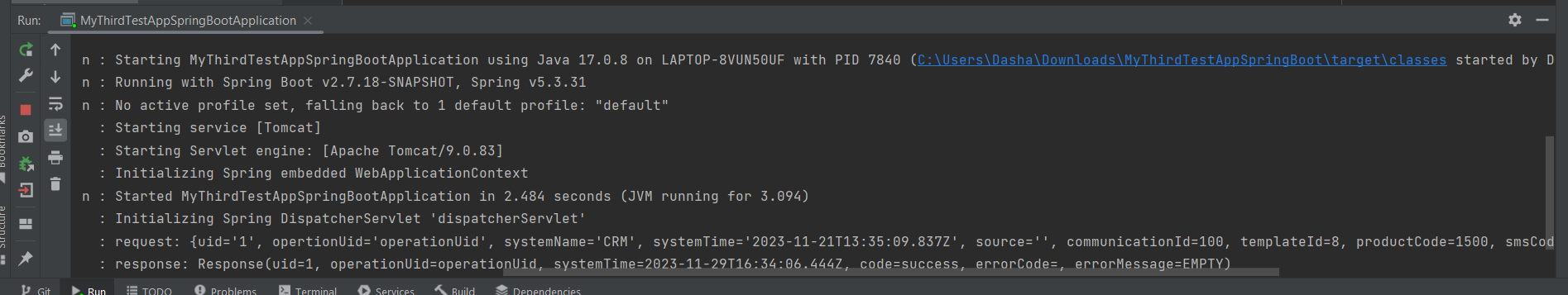
package ru.example.MySecondTestAppSpringBoot.controller;  
  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RestController;  
import ru.example.MySecondTestAppSpringBoot.exception.\*;  
import ru.example.MySecondTestAppSpringBoot.model.\*;  
import ru.example.MySecondTestAppSpringBoot.service.ValidationService;  
import ru.example.MySecondTestAppSpringBoot.service.UnsupportedCodeValidation;  
import ru.example.MySecondTestAppSpringBoot.service.ModifyResponseService;  
import ru.example.MySecondTestAppSpringBoot.util.DateTimeUtil;  
import org.springframework.validation.BindingResult;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Qualifier;  
import javax.validation.Valid;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
import lombok.extern.slf4j.Slf4j;  
  
  
@Slf4j  
@RestController  
public class MyController {  
 private final ValidationService validationService;  
 private final ModifyResponseService modifyResponseService;  
 private final ModifyRequestService modifyRequestService;  
 private final UnsupportedCodeValidation unsupportedCodeValidation;  
  
 @Autowired  
 public MyController(ValidationService validationService, UnsupportedCodeValidation unsupportedCodeValidation,  
 @Qualifier("ModifySystemTimeResponseService") ModifyResponseService modifyResponseService,  
 ModifyRequestService modifyRequestService) {  
 this.validationService = validationService;  
 this.unsupportedCodeValidation = unsupportedCodeValidation;  
 this.modifyResponseService = modifyResponseService;  
 this.modifyRequestService = modifyRequestService;  
 }  
  
 @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);  
 log.info("response: {}", response);  
 return new ResponseEntity<>(modifyResponseService.modify(response), HttpStatus.*OK*);  
 }  
}}

Проверка работоспособности, запрос к серверу 1:

Логи Сервиса 1



Логи Сервиса 2 с модифицированным в Сервисе 1 request:



1. **Реализация дополнительного функционала**
2. Добавьте реализацию интерфейса ModifyRequestService, которая изменяет поле source в Сервисе 1.

Доработка класса ModifySystemNameRequestService:

import org.springframework.core.ParameterizedTypeReference;  
import org.springframework.http.HttpEntity;  
import org.springframework.http.HttpMethod;  
import org.springframework.stereotype.Service;  
import org.springframework.web.client.RestTemplate;  
import ru.example.MySecondTestAppSpringBoot.model.Request;  
import ru.example.MySecondTestAppSpringBoot.model.Systems;  
  
@Service  
public class ModifySystemNameRequestService implements ModifyRequestService {  
 @Override  
 public void modify(Request request) {  
 request.setSystemName(Systems.*CRM*);  
 HttpEntity<Request> httpEntity = new HttpEntity<>(request);  
 new RestTemplate().exchange("http://localhost:8084/feedback",  
 HttpMethod.*POST*,  
 httpEntity,  
 new ParameterizedTypeReference<>() {  
 });  
 }  
}

1. Доработайте Сервис 1 и Сервис 2 таким образом, чтобы в логи Сервиса 2 выводилось сообщение о разнице времени прошедшего с момента получения Сервисом 1 Request и временем получения модифицированного Request полученного Сервисом 2, другими словами, измерить время от нажатия кнопки Send в Postman до вывода в лог Сервисом 2 сообщения.

Request

1. 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;  
     
    @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 +  
    '}';  
    }  
   }

**Ответы на вопросы**

2 вариант

2. Зачем нужна аннотация @SpringBootApplication?

Аннотация @SpringBootApplication используется для объединения нескольких аннотаций в одну. Она включает в себя аннотации @Configuration, @EnableAutoConfiguration и @ComponentScan.

* @Configuration указывает, что класс содержит настройки для контекста приложения.
* @EnableAutoConfiguration позволяет Spring Boot автоматически настраивать компоненты и бины в зависимости от классов, которые находятся в classpath.
* @ComponentScan указывает Spring на то, где нужно искать компоненты и бины, которые нужно зарегистрировать в контексте приложения.

4. Зачем нужна аннотация @NoArgsConstructor?

Аннотация @NoArgsConstructor используется для автоматической генерации конструктора без параметров.

Эта аннотация может быть особенно полезна в контексте использования библиотек, которые требуют наличия конструктора без параметров, например, для десериализации объектов из JSON или XML.

6. Зачем нужна аннотация @Data?

Аннотация @Data в Java используется для автоматической генерации методов геттеров, сеттеров, методов equals, hashCode и toString для всех полей класса. Кроме того, аннотация @Data позволяет создавать объекты класса с помощью конструктора без параметров (если он есть) и упрощает работу с коллекциями.

**Ссылка на репозиторий git**:

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