|  |  |
| --- | --- |
|  | 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 № 1

Руководитель ст. пр. Н. А. Архипов

Студент гр. РИМ-230971 Е.В. Треглазов

Екатеринбург 2024

Цель работы: разработать простой REST сервис.

Реализация простого REST сервиса с валидацией a.

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.model.Request;  
import ru.example.MySecondTestAppSpringBoot.model.Response;  
import ru.example.MySecondTestAppSpringBoot.service.ValidationService;  
import ru.example.MySecondTestAppSpringBoot.exception.ValidationFailedException;  
import ru.example.MySecondTestAppSpringBoot.exception.UnsupportedCodeException;  
import ru.example.MySecondTestAppSpringBoot.service.UnsupportedCodeValidation;  
import org.springframework.validation.BindingResult;  
import org.springframework.beans.factory.annotation.Autowired;  
import javax.validation.Valid;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
@RestController  
public class MyController {  
 private final ValidationService validationService;  
 private final UnsupportedCodeValidation unsupportedCodeValidation;  
  
 @Autowired  
 public MyController(ValidationService validationService, UnsupportedCodeValidation unsupportedCodeValidation) {  
 this.validationService = validationService;  
 this.unsupportedCodeValidation = unsupportedCodeValidation;  
 }  
  
 @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'");  
  
 Response response = Response.builder()  
 .uid(request.getUid())  
 .operationUid(request.getOperationUid())  
 .systemTime(simpleDateFormat.format(new Date()))  
 .code("success")  
 .errorCode("")  
 .errorMessage("")  
 .build();  
  
 try {  
 validationService.isValid(bindingResult);  
 } catch (ValidationFailedException e) {  
 response.setCode("failed");  
 response.setErrorCode("ValidationException");  
 response.setErrorMessage("Ошибка валидации");  
 return new ResponseEntity<>(response, HttpStatus.*BAD\_REQUEST*);  
 } catch (Exception e) {  
 response.setCode("failed");  
 response.setErrorCode("UnknownException");  
 response.setErrorMessage("Произошла непредвиденная ошибка");  
 return new ResponseEntity<>(response, HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
  
 try {  
 unsupportedCodeValidation.Valid(Integer.*parseInt*(request.getUid()));  
 } catch (UnsupportedCodeException e) {  
 response.setCode("failed");  
 response.setErrorCode("UnsupportedCodeException");  
 response.setErrorMessage("uid = 123");  
 return new ResponseEntity<>(response, HttpStatus.*BAD\_REQUEST*);  
 }  
  
 return new ResponseEntity<>(response, HttpStatus.*OK*);  
 }  
}

UnsupportedCodeException

package ru.example.MySecondTestAppSpringBoot.exception;  
  
public class UnsupportedCodeException extends Exception{  
 public UnsupportedCodeException() {  
 super("uid = 123");  
 }  
}

b. ValidationFailedException

package ru.example.MySecondTestAppSpringBoot.exception;  
  
public class ValidationFailedException extends Exception {  
 public ValidationFailedException(String message) {super(message);}  
}

c. 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 String systemName;  
  
 @NotBlank  
 private String systemTime;  
  
 private String source;  
  
 @Min(1)  
 @Max(100000)  
 private int communicationId;  
  
 private int templateId;  
  
 private int productCode;  
  
 private int smsCode;  
}

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 String errorCode;  
 private String errorMessage;  
}

RequestValidationService

package ru.example.MySecondTestAppSpringBoot.service;  
  
import org.springframework.stereotype.Service;  
import org.springframework.validation.BindingResult;  
import ru.example.MySecondTestAppSpringBoot.exception.ValidationFailedException;  
  
@Service  
public class RequestValidationService implements ValidationService {  
 @Override  
 public void isValid(BindingResult bindingResult) throws ValidationFailedException {  
 if (bindingResult.hasErrors()) {  
 throw new  
 ValidationFailedException(bindingResult.getFieldError().toString());  
 }  
 }  
}

UnsupportedCodeValidation

package ru.example.MySecondTestAppSpringBoot.service;  
  
import org.springframework.stereotype.Service;  
import ru.example.MySecondTestAppSpringBoot.exception.UnsupportedCodeException;  
  
@Service  
public class UnsupportedCodeValidation {  
 public void Valid(int uid) throws UnsupportedCodeException {  
 if (uid == 123) {  
 throw new UnsupportedCodeException();  
 }  
 }  
}

ValidationService

package ru.example.MySecondTestAppSpringBoot.service;  
  
import org.springframework.stereotype.Service;  
import org.springframework.validation.BindingResult;  
import ru.example.MySecondTestAppSpringBoot.exception.ValidationFailedException;  
  
  
@Service  
public interface ValidationService {  
 void isValid(BindingResult bindingResult) throws ValidationFailedException;  
}

1 Добавьте валидацию полей в соответствии с п 1.1 в таблице с описанием request,

колонка обязательность.

public class Request {  
 @NotBlank  
 @Size(max = 32)  
 private String uid;  
  
 @NotBlank  
 @Size(max = 32)  
 private String operationUid;  
  
 private String systemName;  
  
 @NotBlank  
 private String systemTime;  
  
 private String source;  
  
 @Min(1)  
 @Max(100000)  
 private int communicationId;  
  
 private int templateId;  
  
 private int productCode;  
  
 private int smsCode;  
}

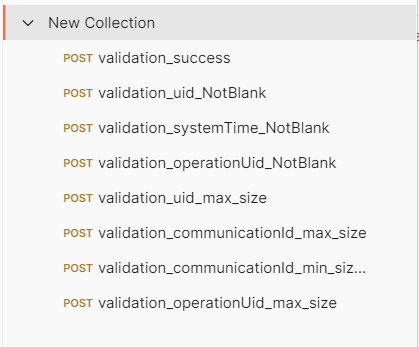
2 Приложите скриншоты тестирования валидации, проверьте все кейсы валидации

поочередно, а затем все вместе, в итоге у вас должно пулучиться несколько тест кейсов,

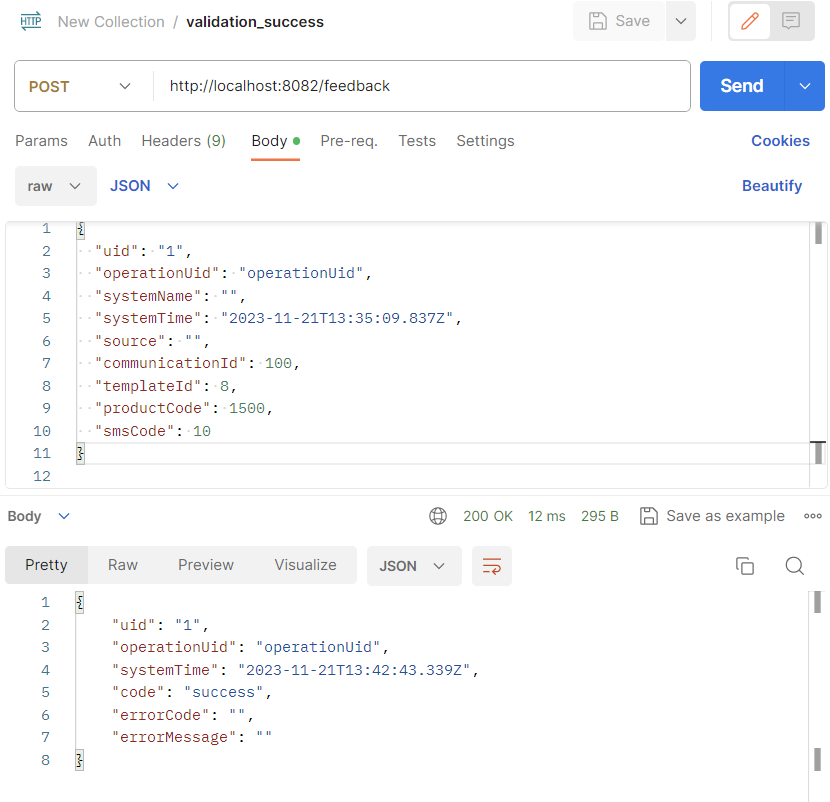
приложите скиншот Postman с созданным каталогом для ЛР и всеми запросами-тест

кейсами.

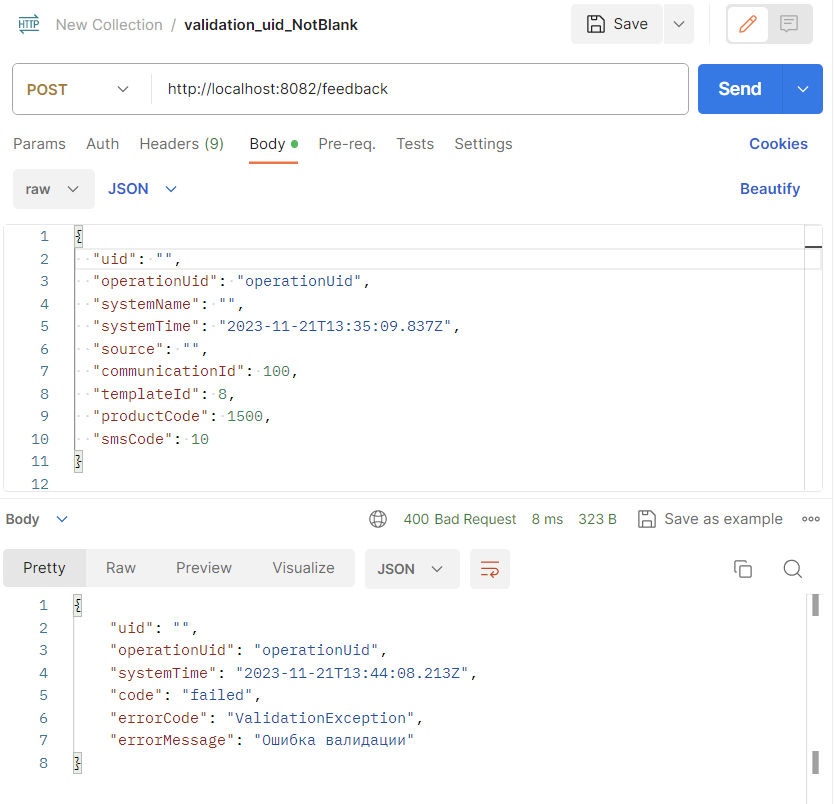
Каталог в Postman:



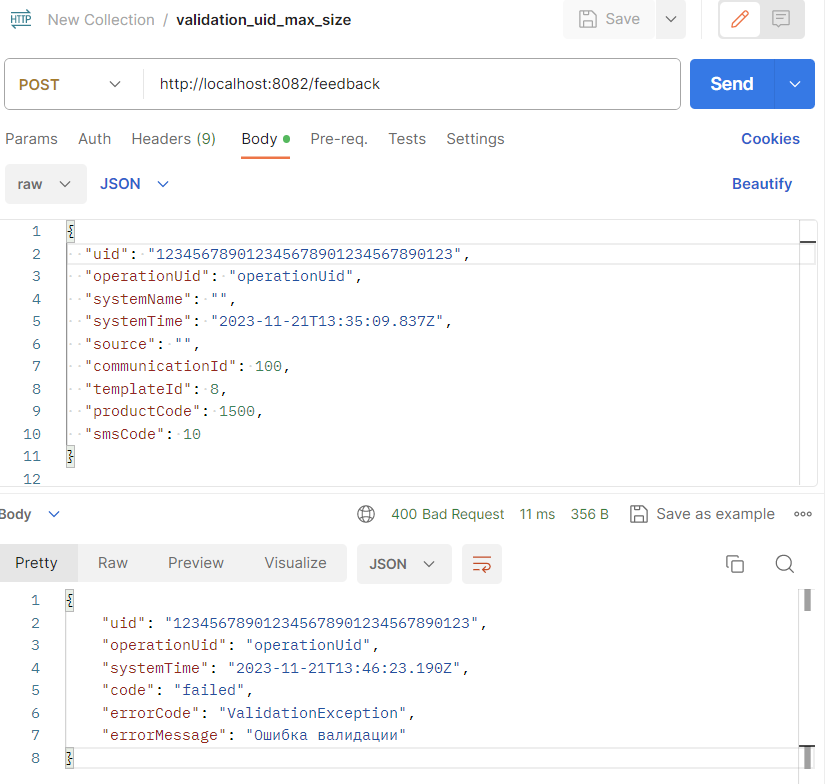
Запрос POST:



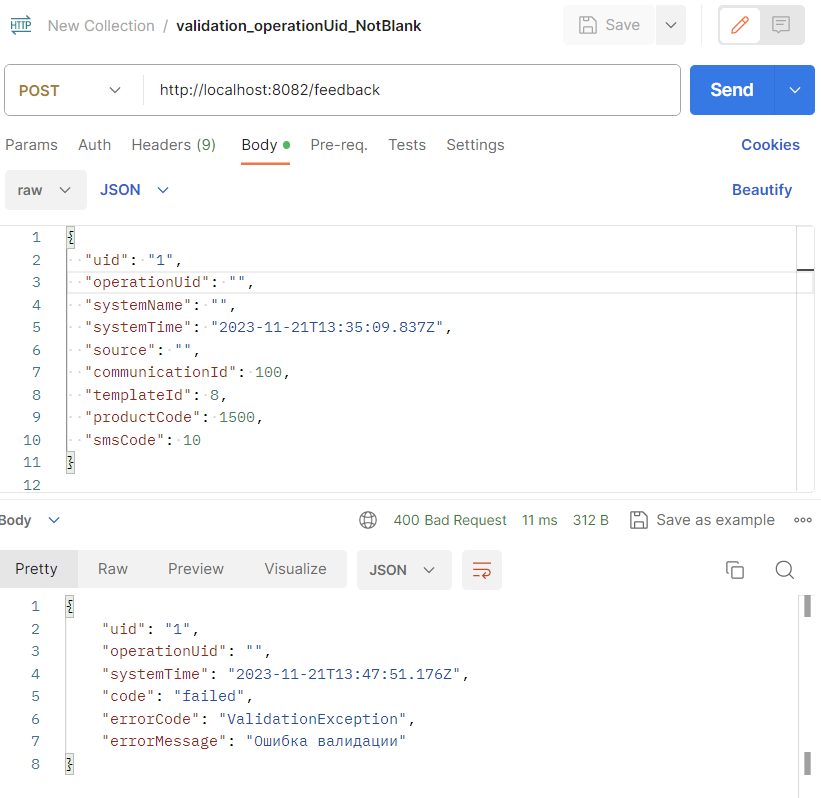
Тест-кейс проверка обязательности uid:



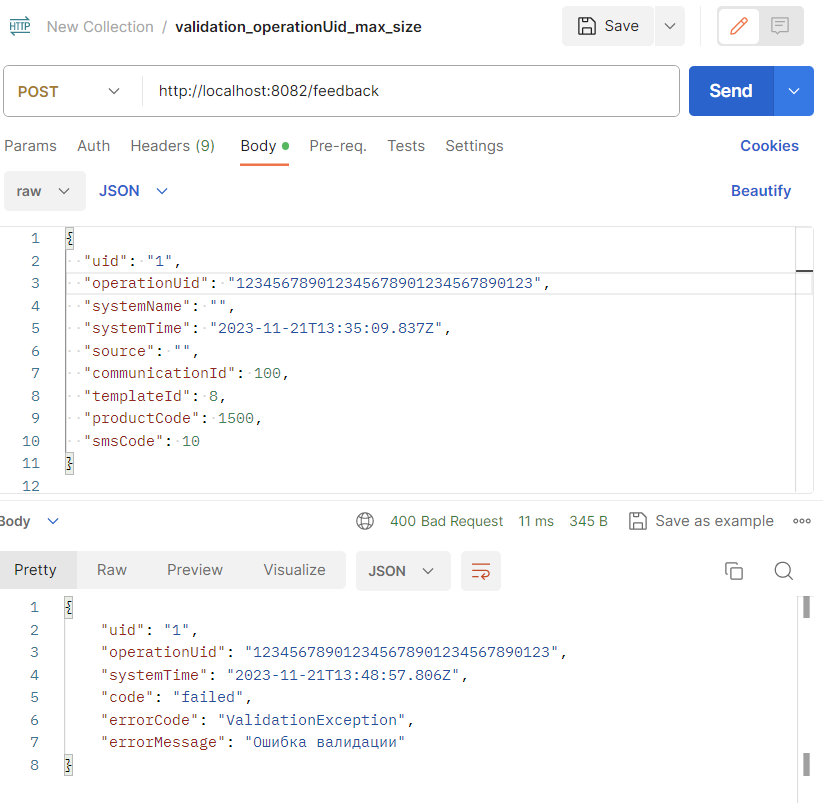
Тест-кейс проверка максимальной длины uid:



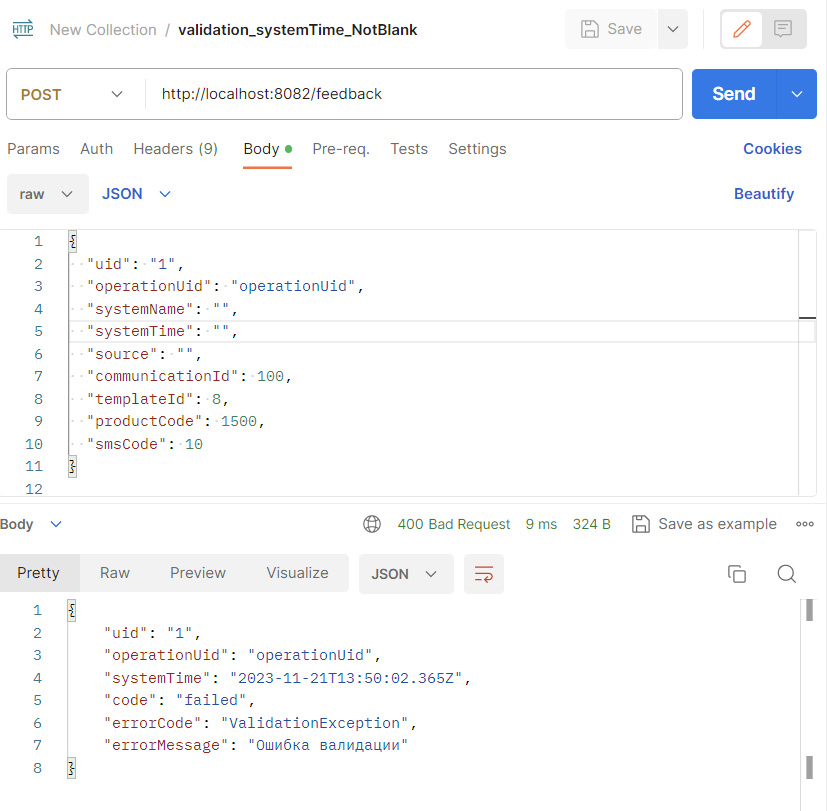
Тест-кейс проверка обязательности operationUid:



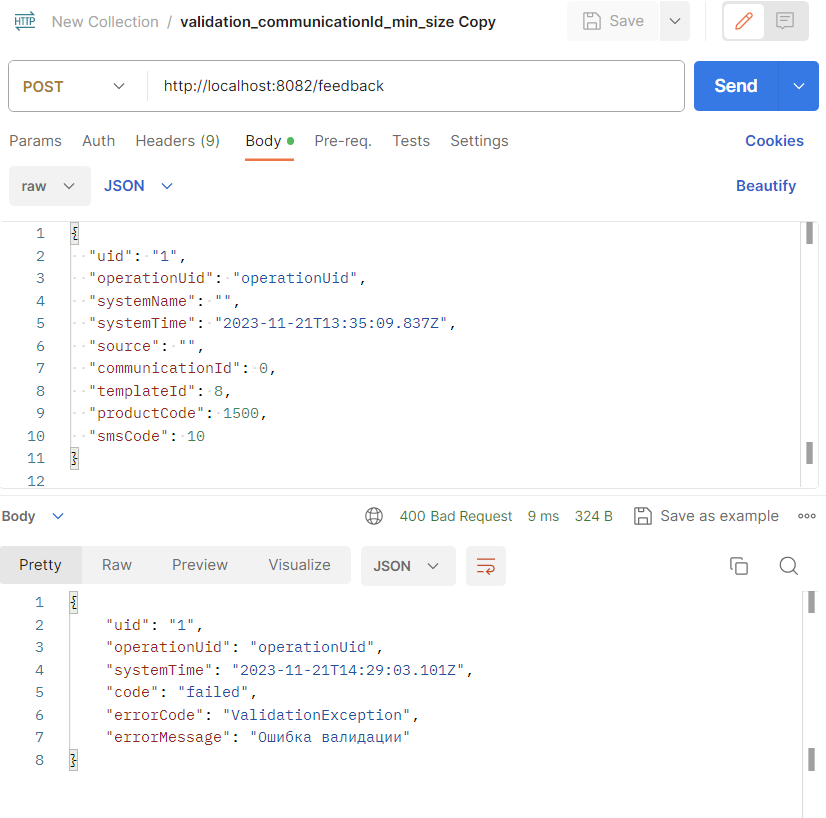
Тест-кейс проверка максимальной длины operationUid:



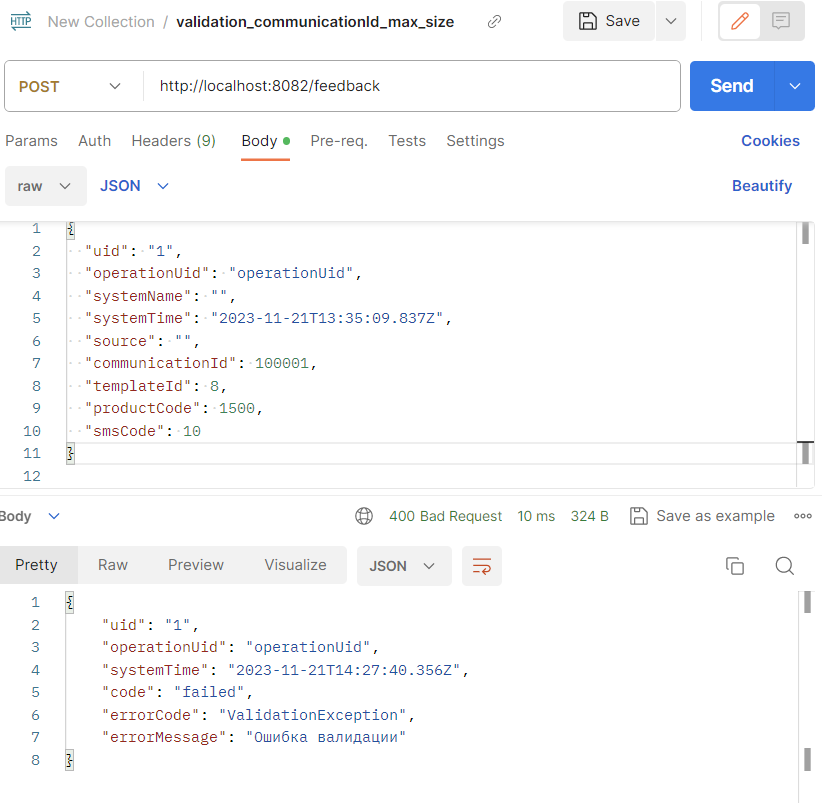
Тест-кейс проверка обязательности systemTime:



Тест-кейс проверка минимальной длины communicationId:

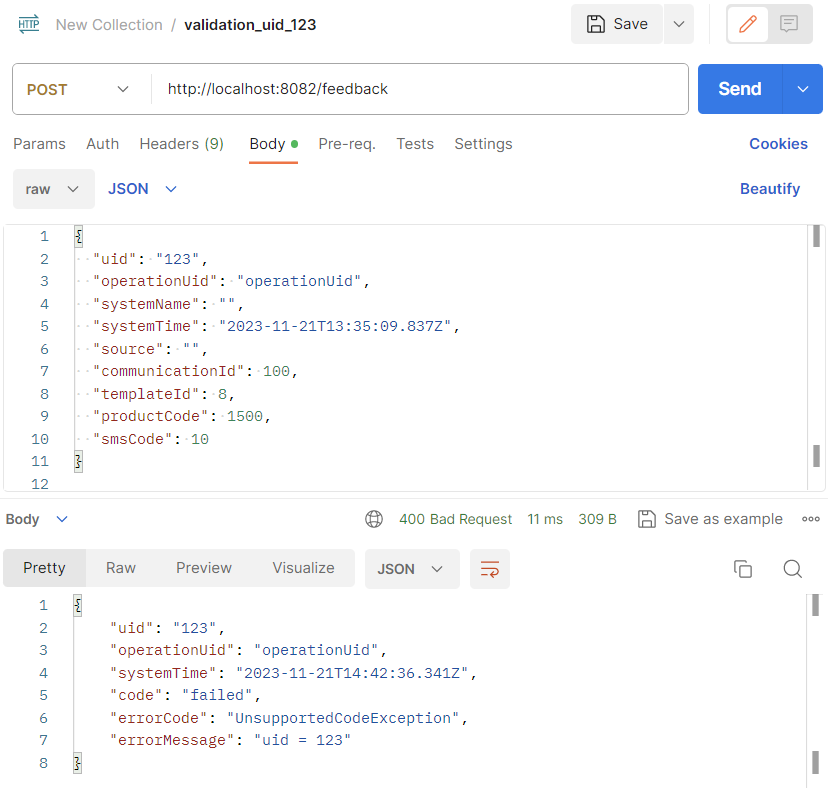


Тест-кейс проверка максимальной длины communicationId:



4. Создайте собственное исключение UnsupportedCodeException, выбрасывайте его в случае, когда uid равен 123, добавьте код ошибки в ответное сообщение, приложите скриншот выполненного теста.

Тест-кейс проверка uid=123:



1. **Ответы на вопросы**

2 вариант

2. Какое содержимое может передаваться POST запросом?

* Текстовые данные
* Файлы
* Структурированные данные (например, JSON, XML)
* Бинарные данные

4. Зачем нужны кастомные исключения?

Кастомные исключения нужны для того, чтобы создавать и использовать собственные типы исключительных ситуаций, которые могут возникать в коде.

6. Что означает аннотация @RestController?

Аннотация @RestController используется для обозначения класса как контроллера, который обрабатывает HTTP-запросы и возвращает данные в формате JSON или XML.

**Ссылка на репозиторий git:** https://github.com/kuk86kuk/java\_ufru\_2024