**Министерство науки и высшего образования 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

Центр ускоренного обучения

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**

**По дисциплине «Технологии программирования»**

Студент группы РИЗ-240938у Бабенко М.А.

Преподаватель: Архипов Н.А.

**Екатеринбург, 2025 г.**

**Лабораторная работа N2**

Цель: разработать простой REST сервис.

**Задания для самостоятельной работы**

**Задание 1**

Заходим на сайт <https://start.spring.io/>, формируем конфигурация согласно заданию, и скачиваем архив.

Меню конфигурации показано на рисунке 1.

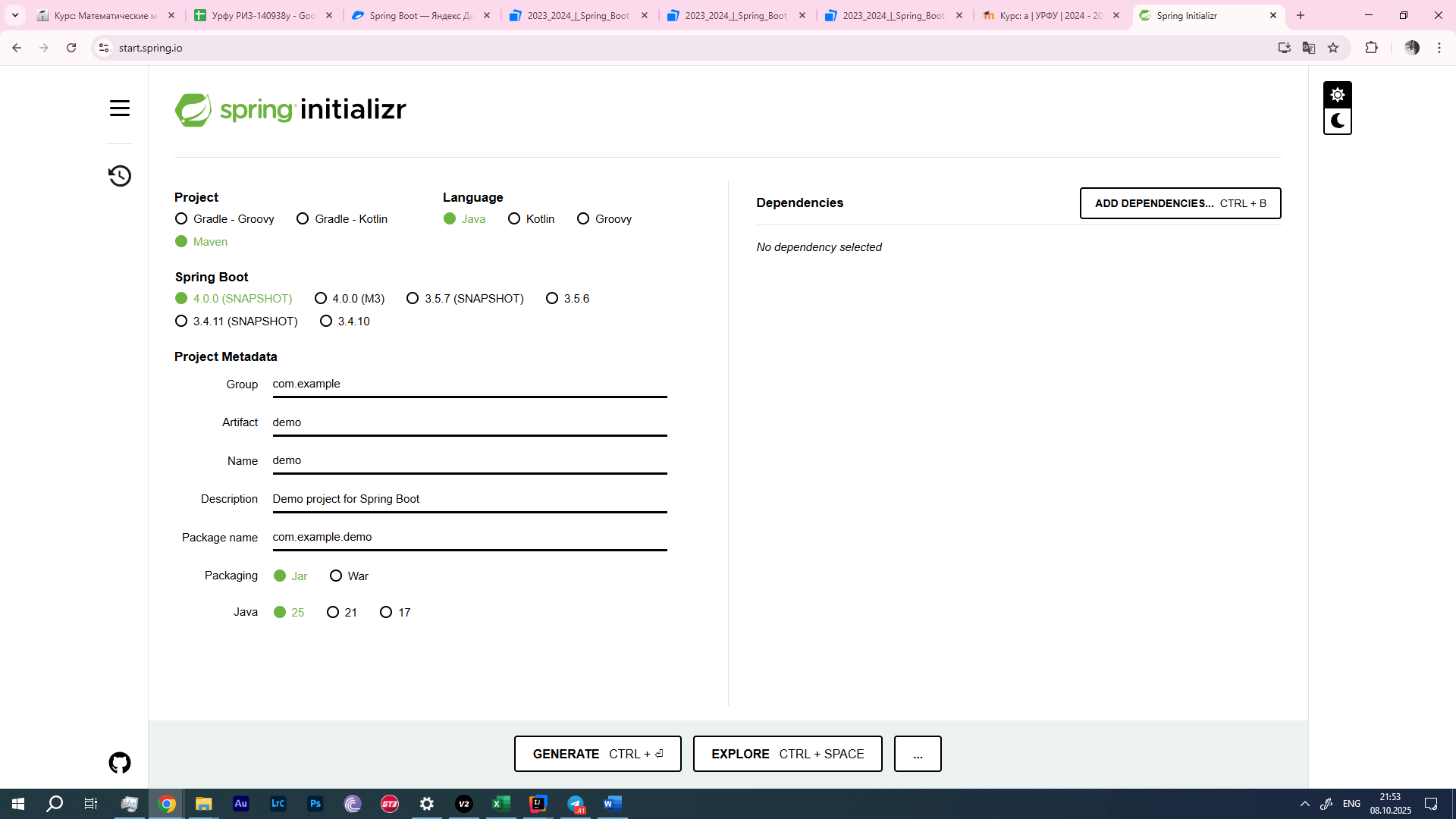


Рисунок 1 – Меню конфигурации

Затем, открываем программу IntelliJ IDEA, содержимое архива, создаем новые пакеты, согласно заданию.

Результат создания новых пакетов показан на рисунке 2.

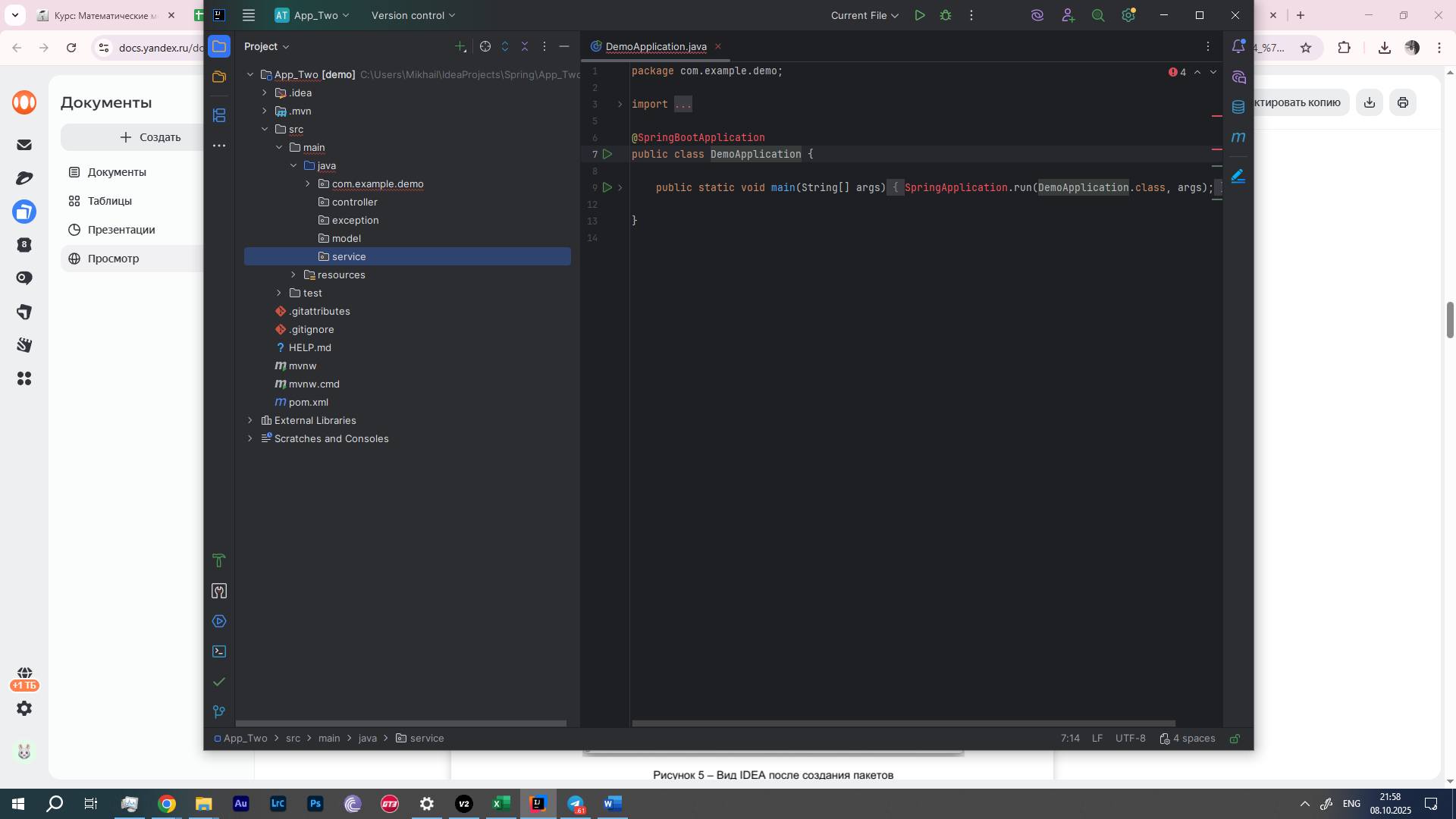


Рисунок 2 – Результат запуска тестовой программы

Затем, создаем еще два класса внутри model, как указано в задании.

Результат создания показан на рисунке 3.

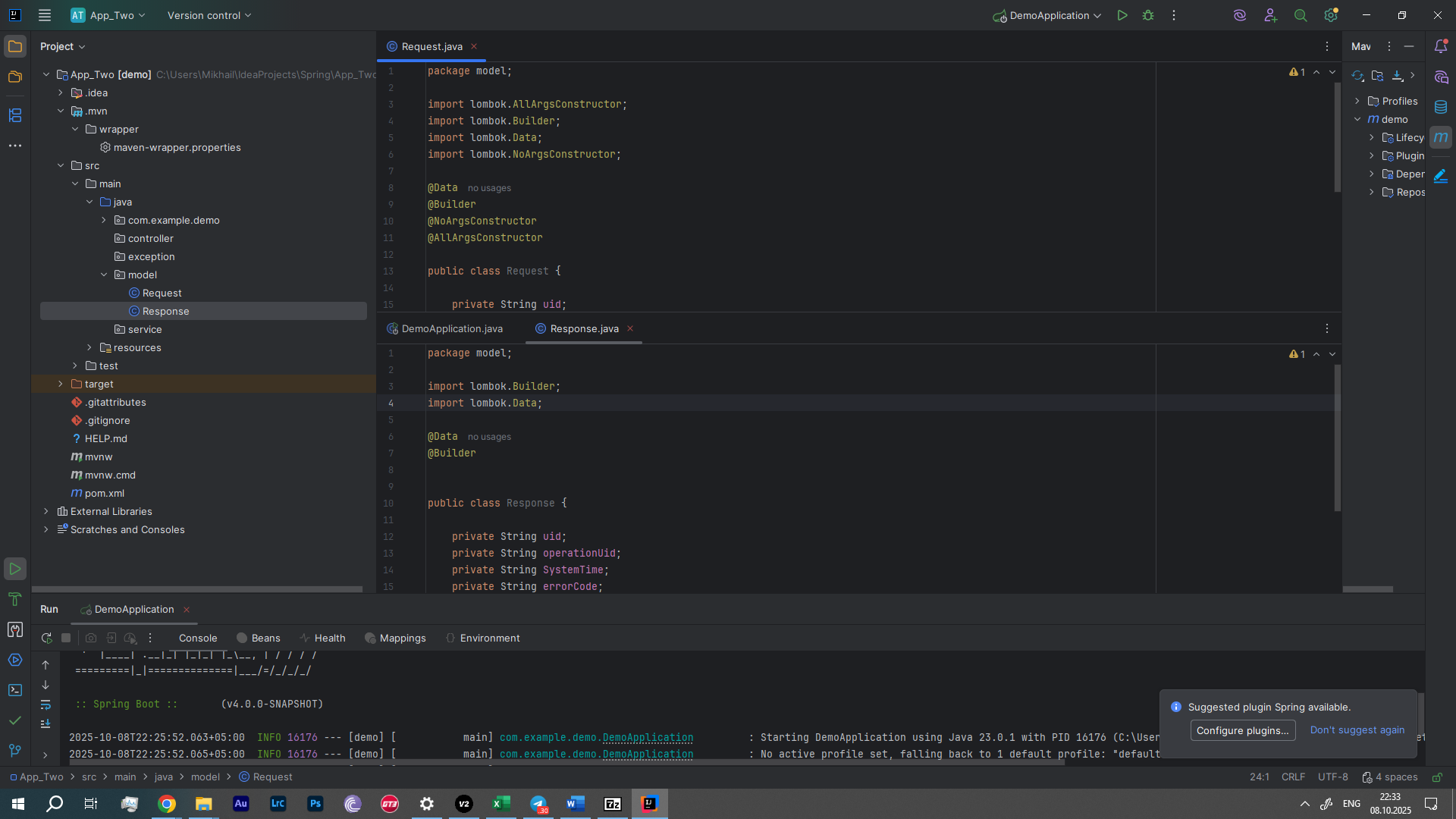


Рисунок 3 –Созданные новые два класса

Далее, создаем класс MyController, в который записываем код программы по аналогии с примером, попутно изменяя его, где это требуется.

Результат создания показан на рисунке 4.

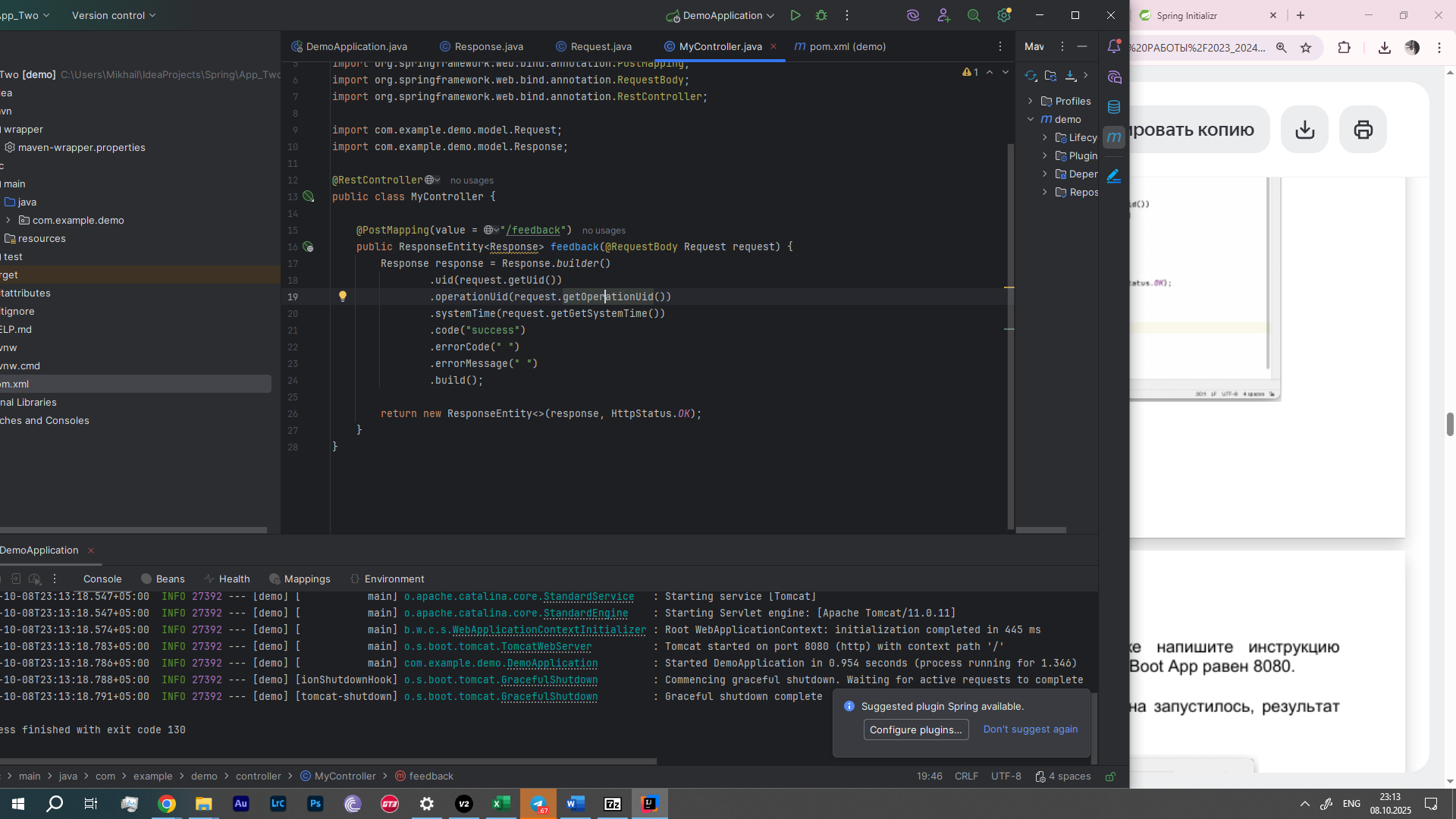


Рисунок 4 – Результат создания класса MyController

Также, требуется изменить порт по умолчанию на 8082 в файле application.properties.



Рисунок 5 – Результат изменения порта по умолчанию

Также создается класс ValidationFailedException по образцу. Его код реализации показан на рисунке 6.

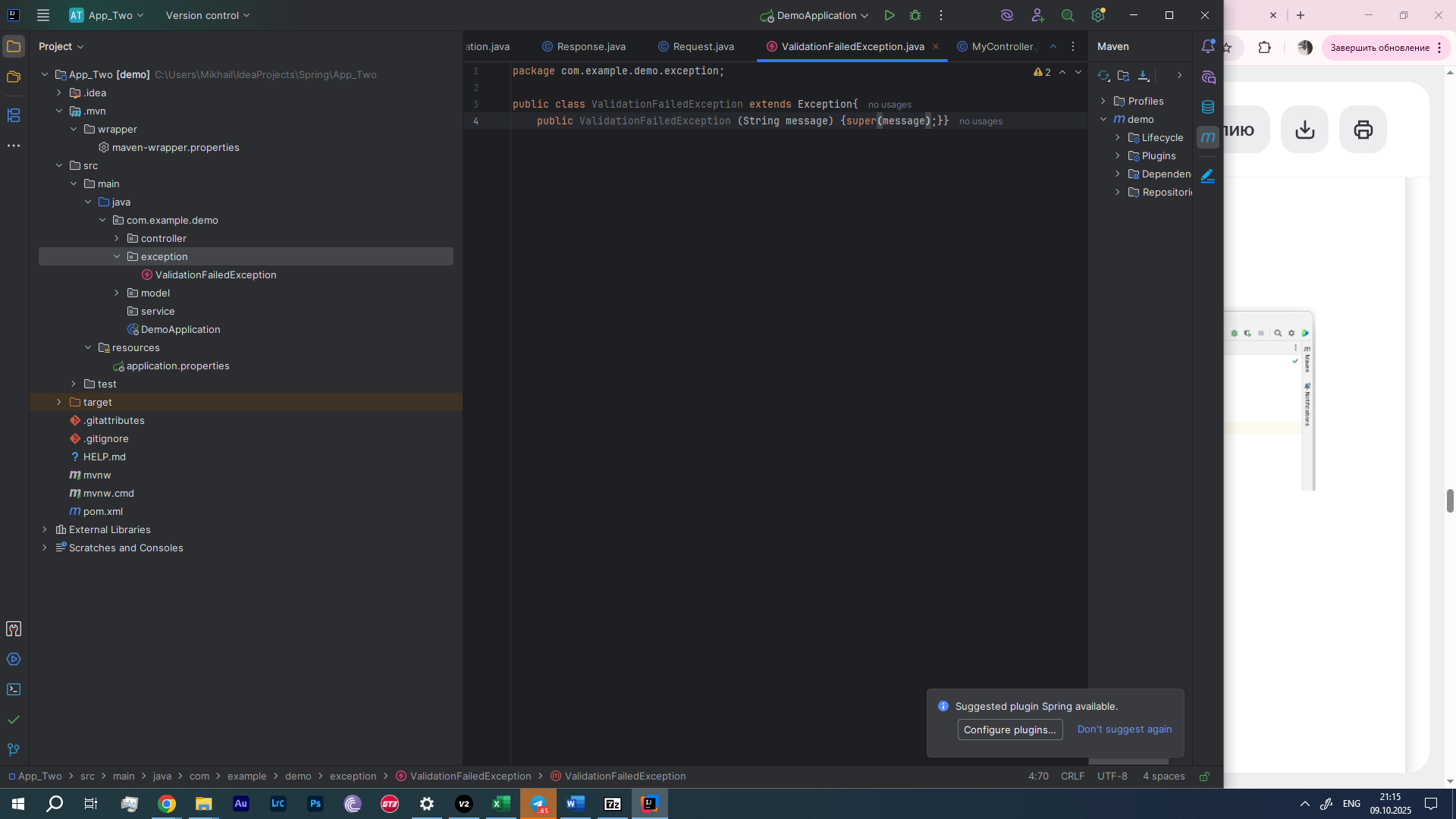


Рисунок 6 – Код реализации класса ValidationFailedException

Дополняем наш главный класс – MyController.java, по образцу. Также исправляем ошибки в других классах, в том числе интерфейсе. Алилуя, компиляция прошла, и программа была запущена.

Дополненный класс MyController.java и процесс успешной компиляции показаны на рисунке 7.

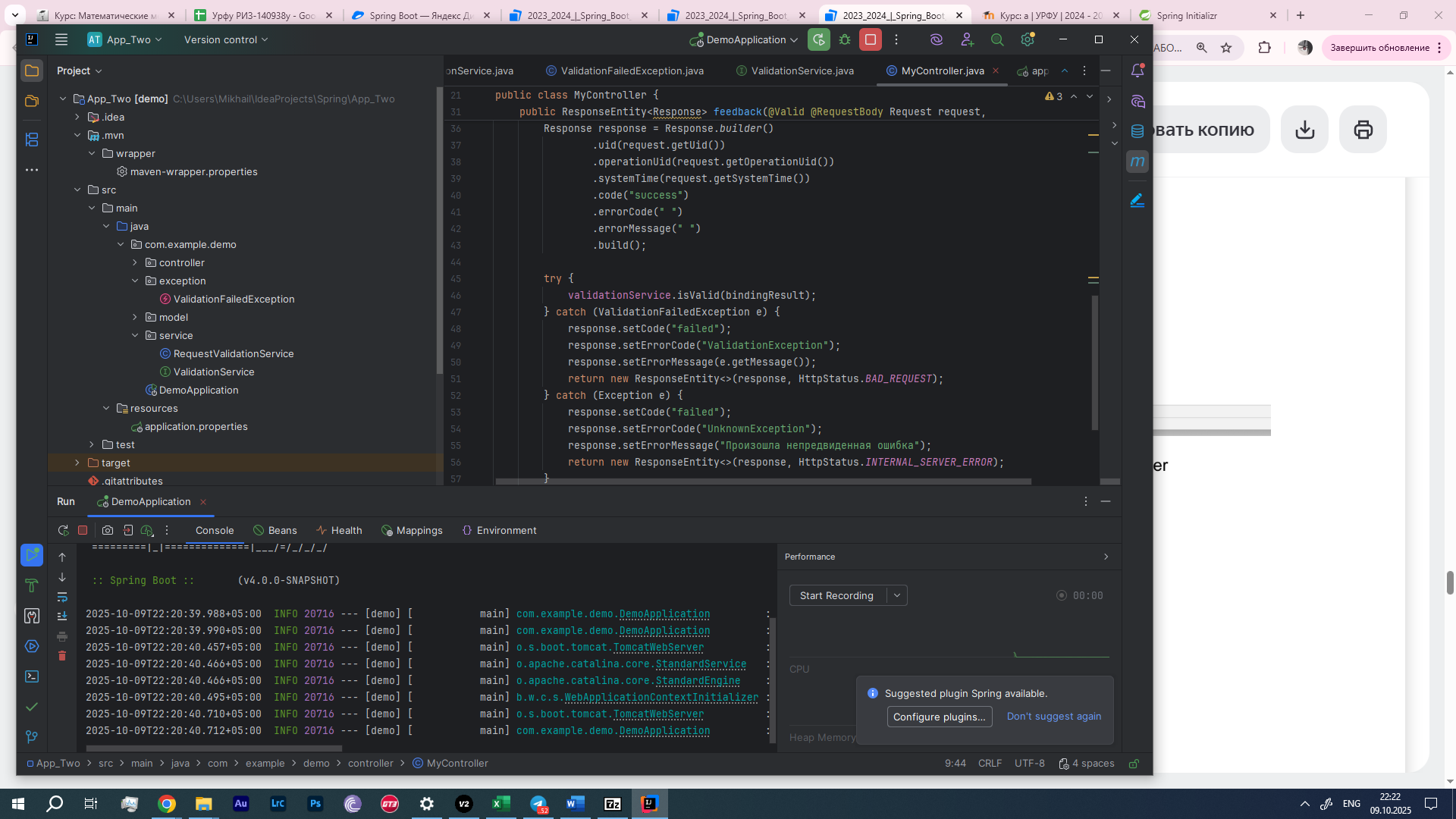


Рисунок 7 – Дополненный MyController.java и процесс успешной компиляции

**Задание 2. Реализация дополнительного функционала.**

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

2 Приложите скриншоты тестирования валидации, проверьте все кейсы валидации поочередно, а затем все вместе, в итоге у вас должно получиться несколько тест кейсов, приложите скиншот Postman с созданным каталогом для ЛР и всеми запросами-тест кейсами;

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

Для начала, дополним метод Request. Добавляем, согласно заданию, валидацию полей. Для этого нам нужно задать предельные значения для полей Request. С помощью @NotBlank и @NotNull задаем правило – строки и переменные не должны быть пустыми, а @Min задает по умолчанию минимальное значение.

Код реализации класса Request показан на рисунке 8.

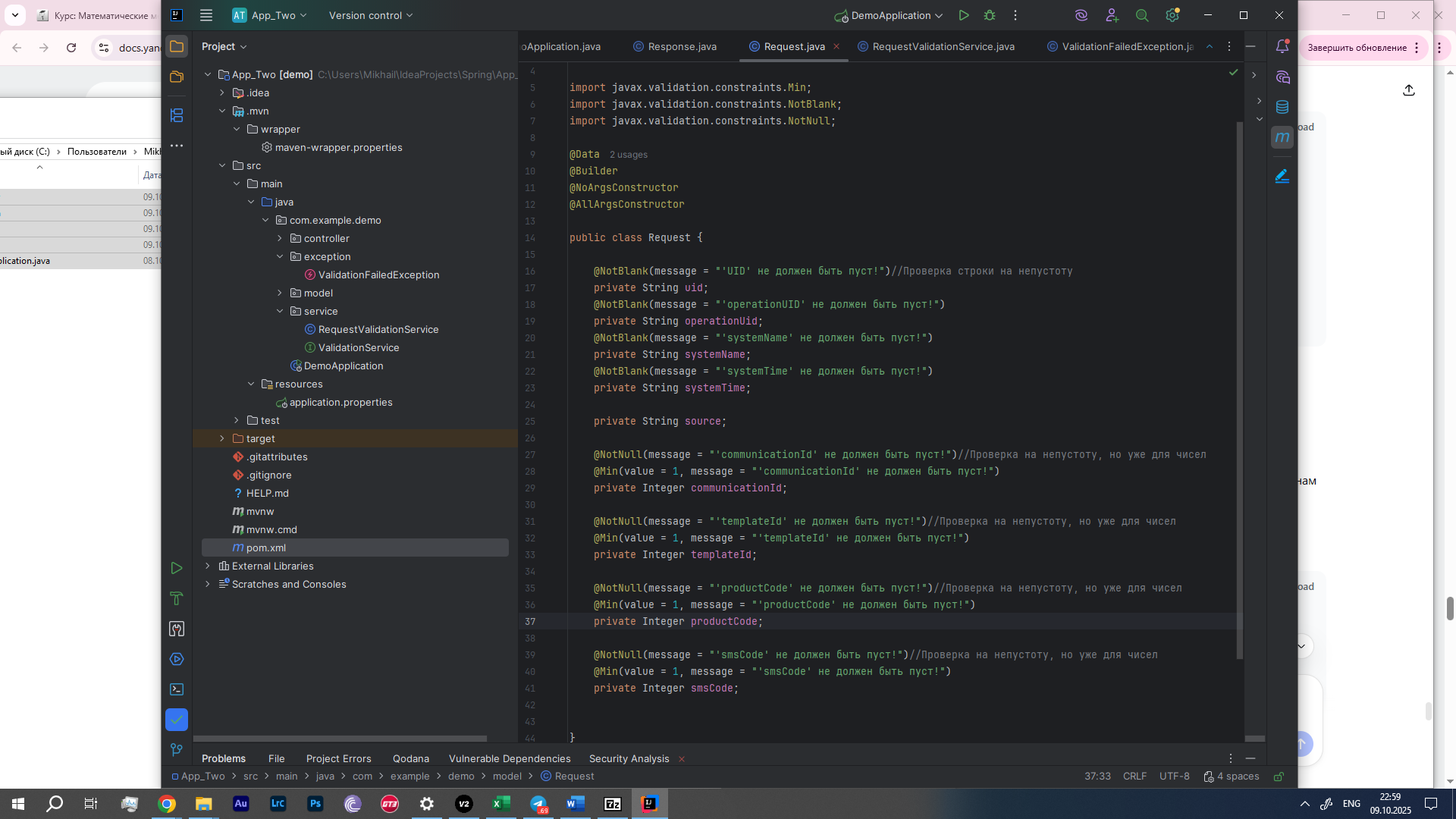


Рисунок 8 – Код реализации класса Request

Также изменим класс RequestValidationService. Добавляем обработчик ошибок валидации, а также реализуем вывод всех возможных ошибок. Код класса показан на рисунке 9.

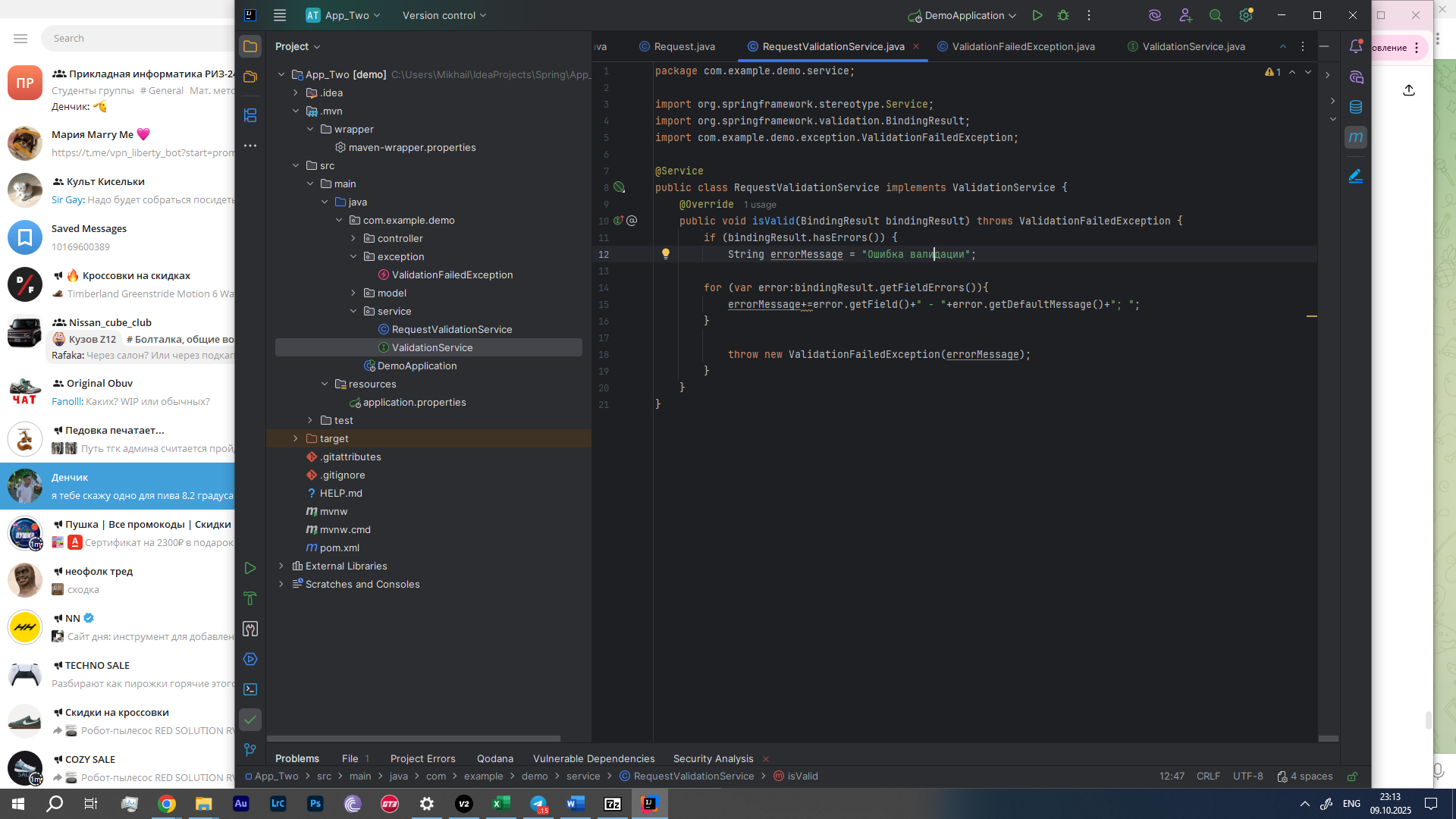


Рисунок 9 – Код класса RequestValidationService

Протестируем нашу программу через Postman. Для проверки валидации, создаем несколько тестов с разными кейсами. Кейс 1 должен быть успешен, остальные 5 – должны завершаться с ошибками. Процессы тестов показаны на рисунках 10…15.

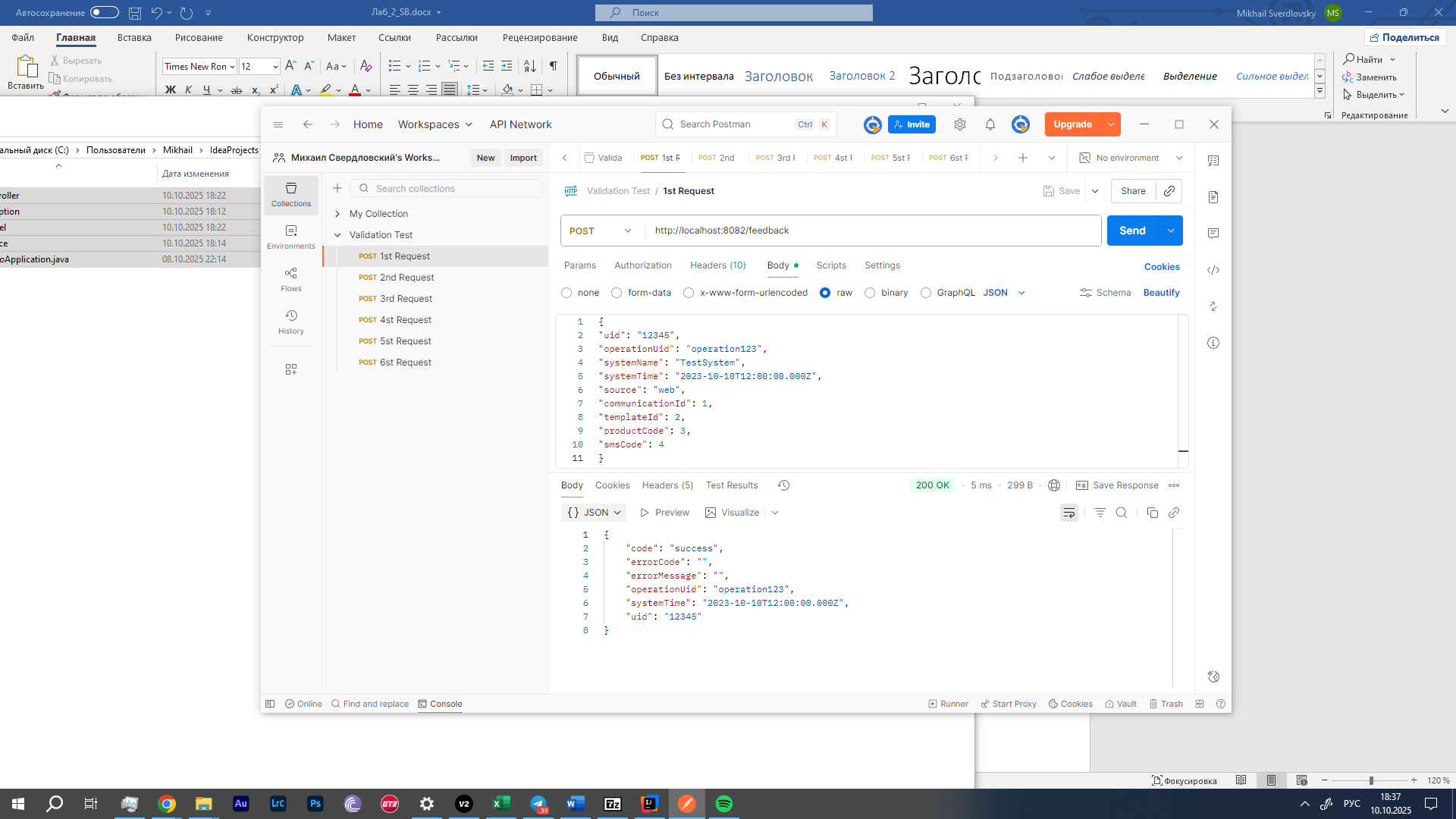


Рисунок 10 – Кейс успешного теста

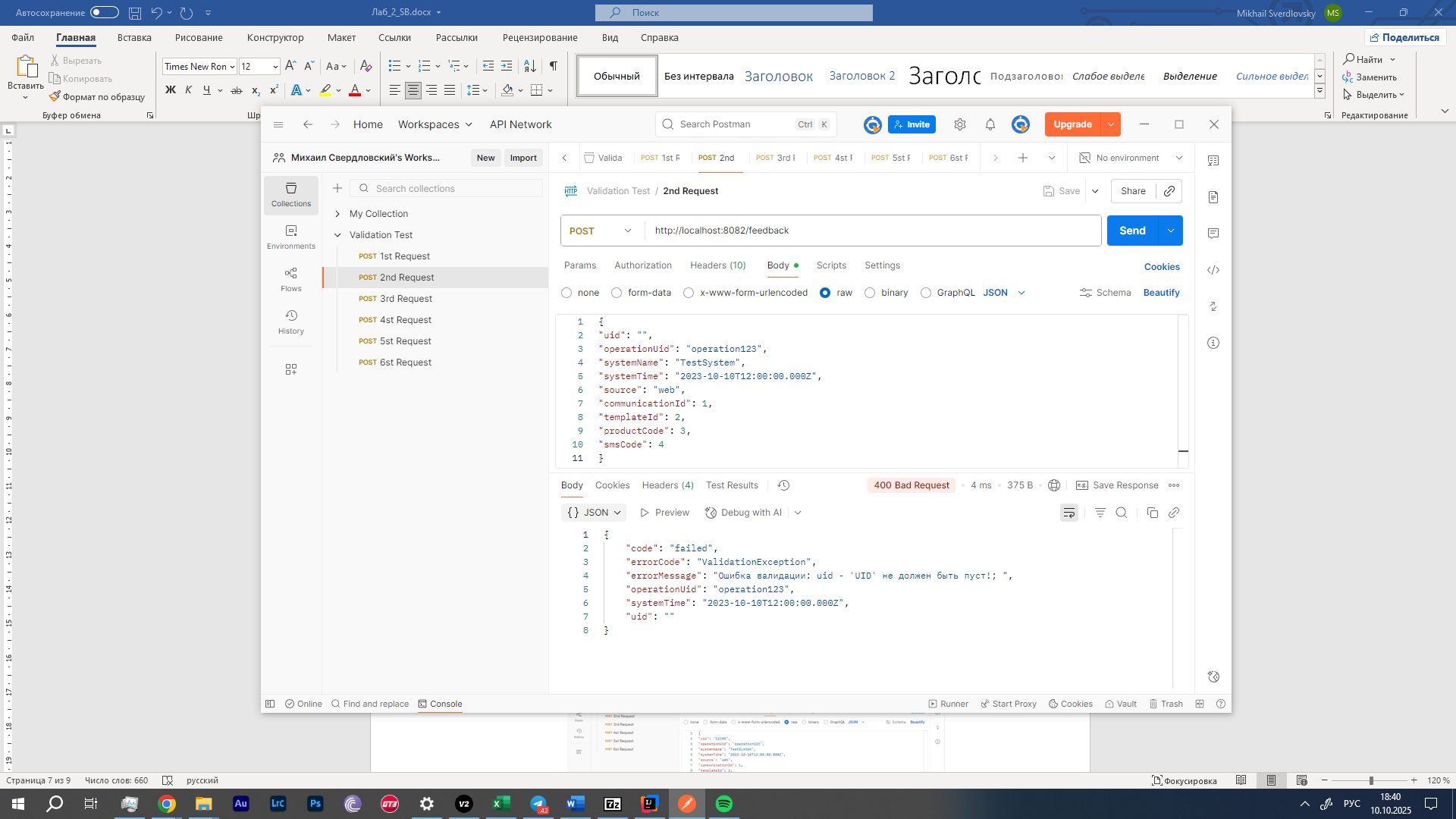


Рисунок 11 – Кейс теста с ошибкой – пустой UID

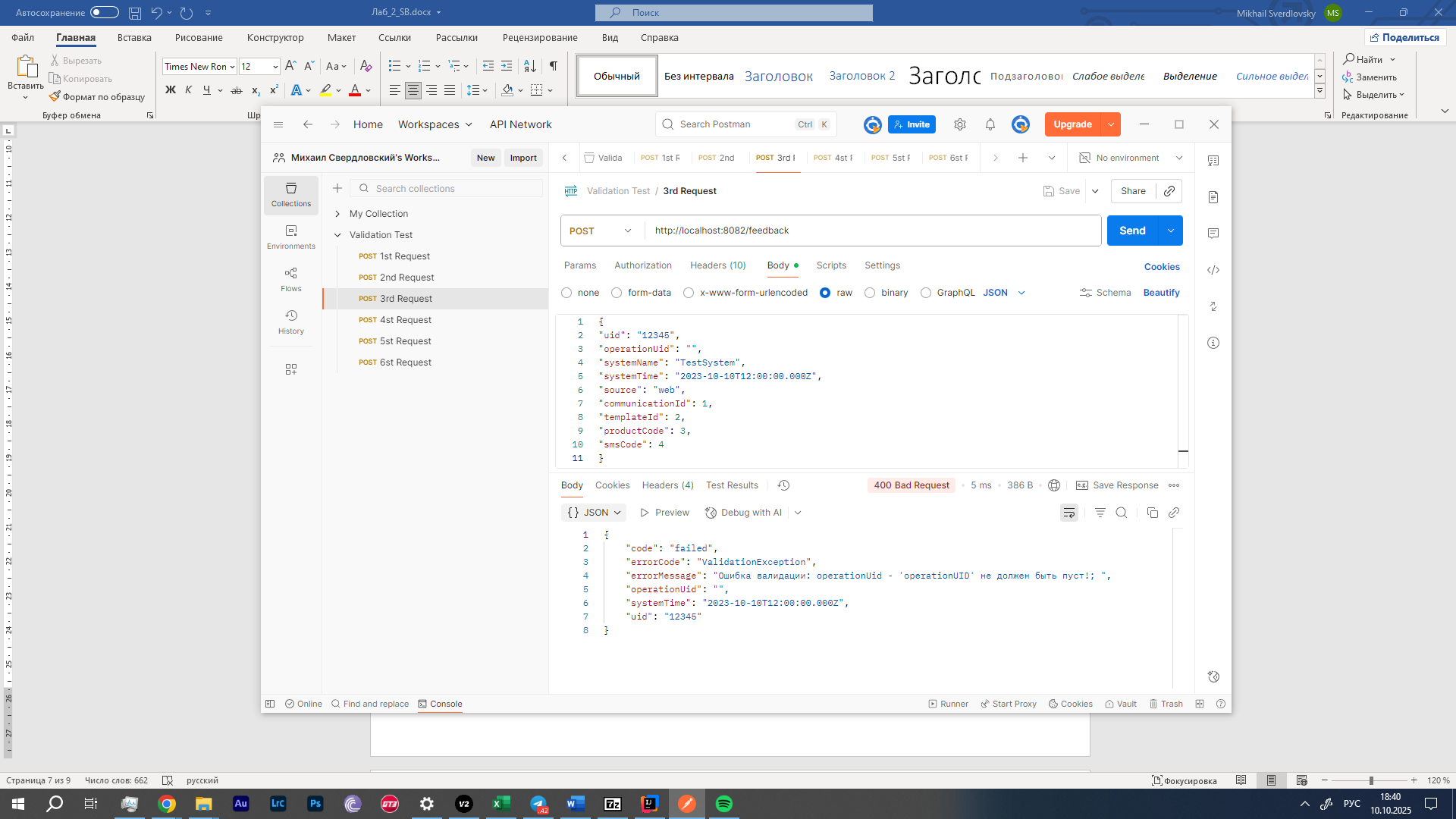


Рисунок 12 – Кейс теста с ошибкой – пустой operationUID

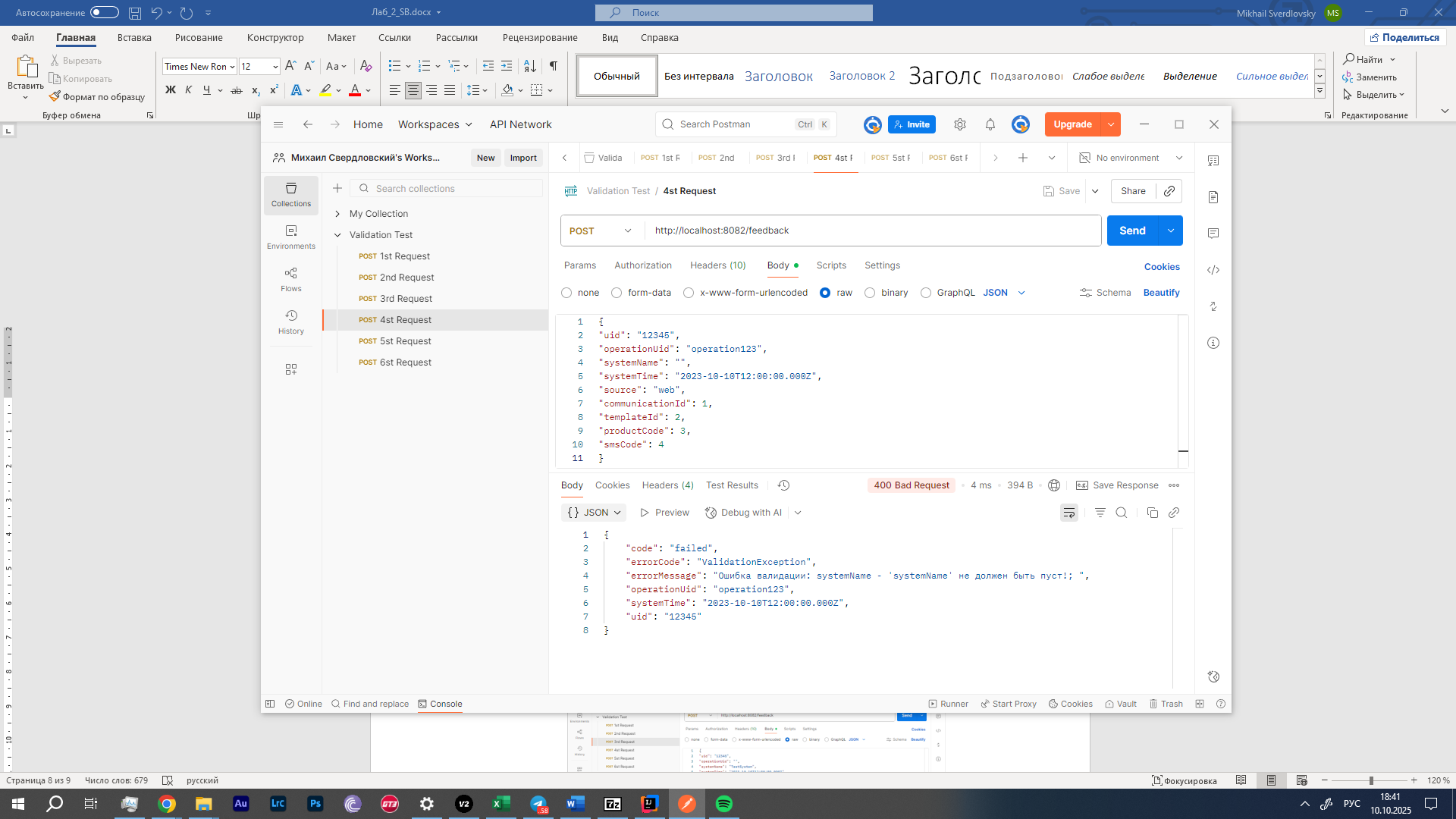


Рисунок 13 – Кейс теста с ошибкой – пустой systemName

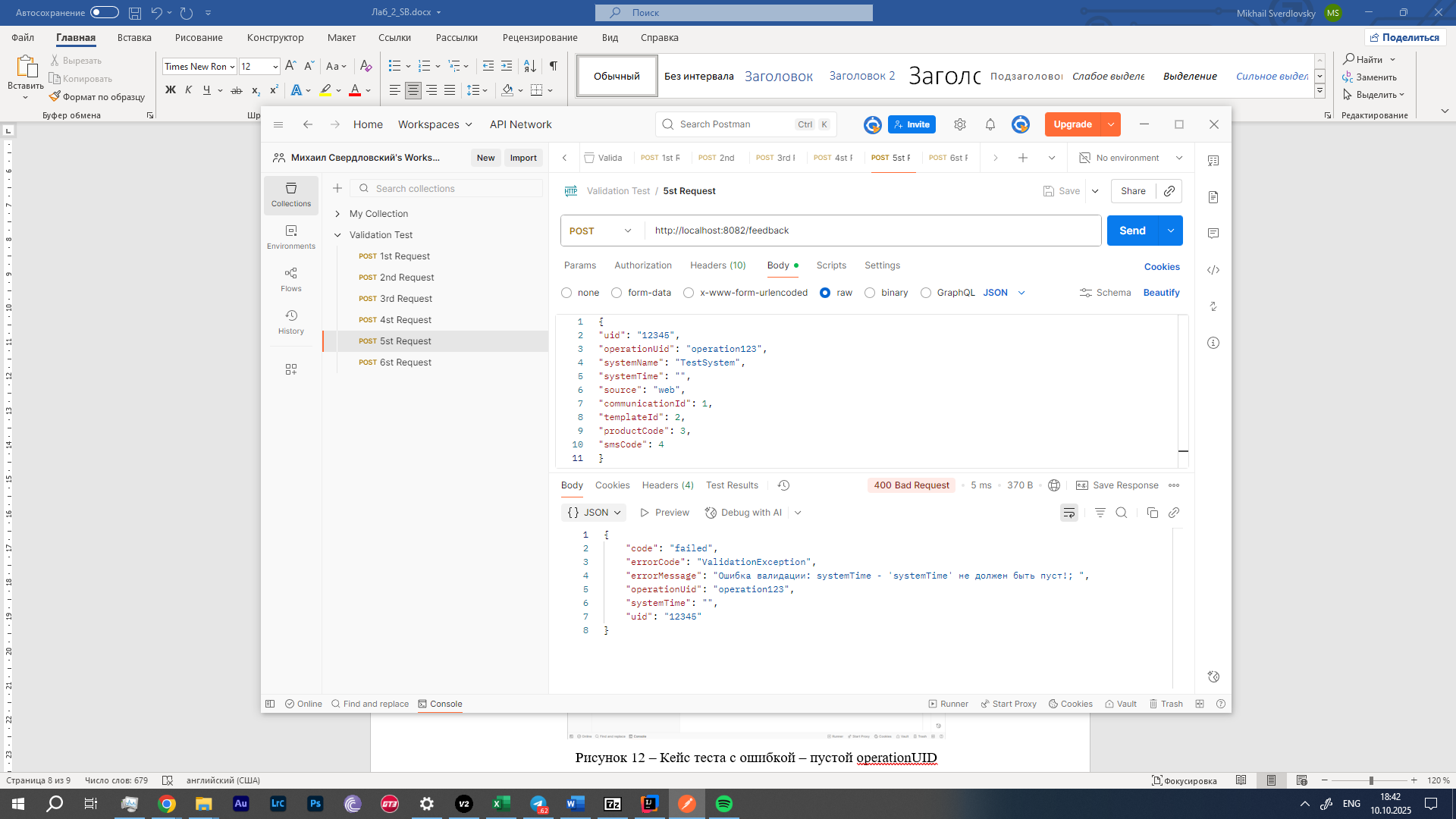


Рисунок 14 – Кейс теста с ошибкой – пустой systemTime

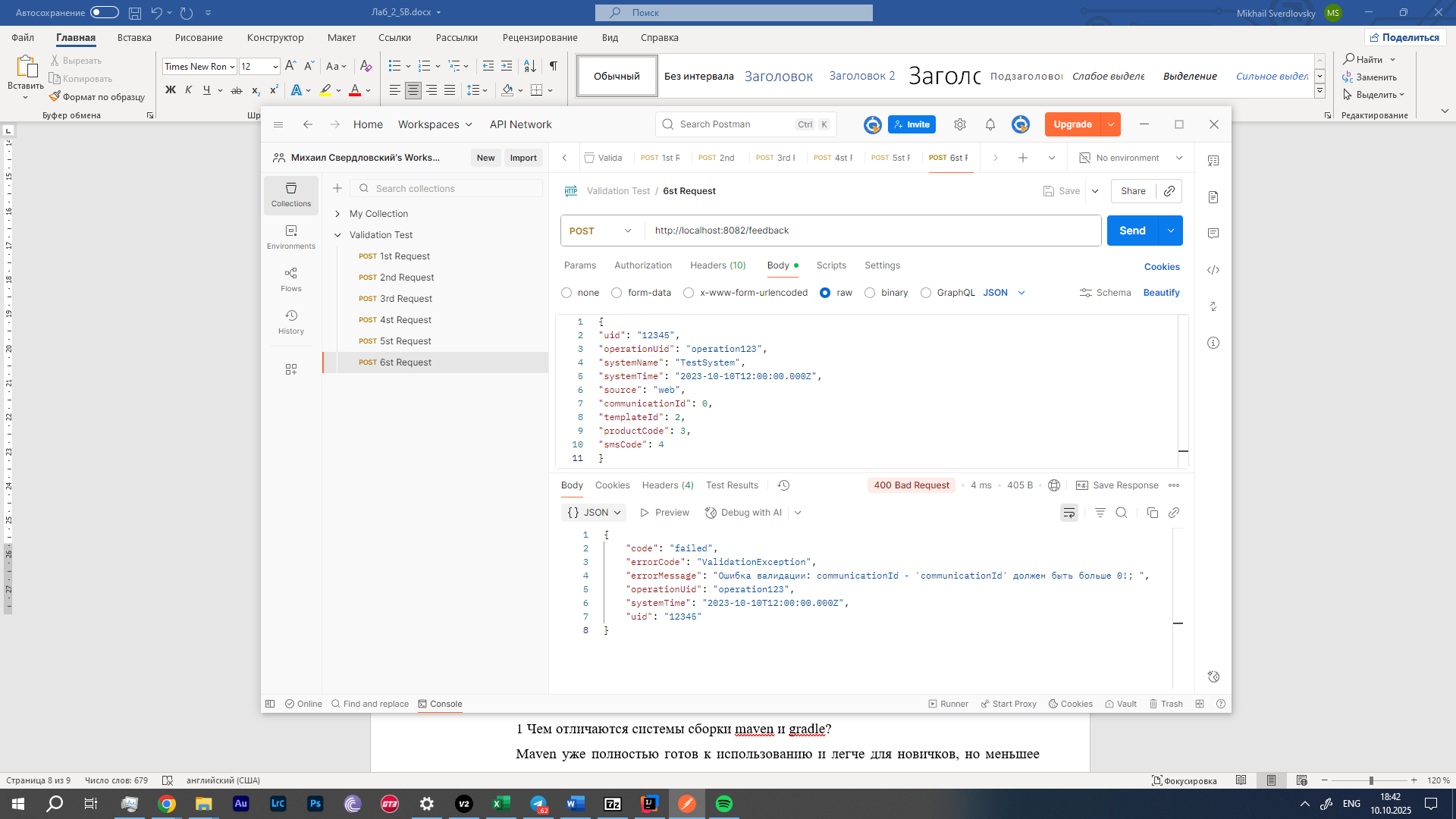


Рисунок 15 – Кейс теста с ошибкой – нулевой communicationID

Согласно заданию, при запросе с UID = 123, должна быть ошибка. Теперь мы добавим возможность такой ошибки. Для этого дополним код класса MyController. Код класса показан на рисунке 16.

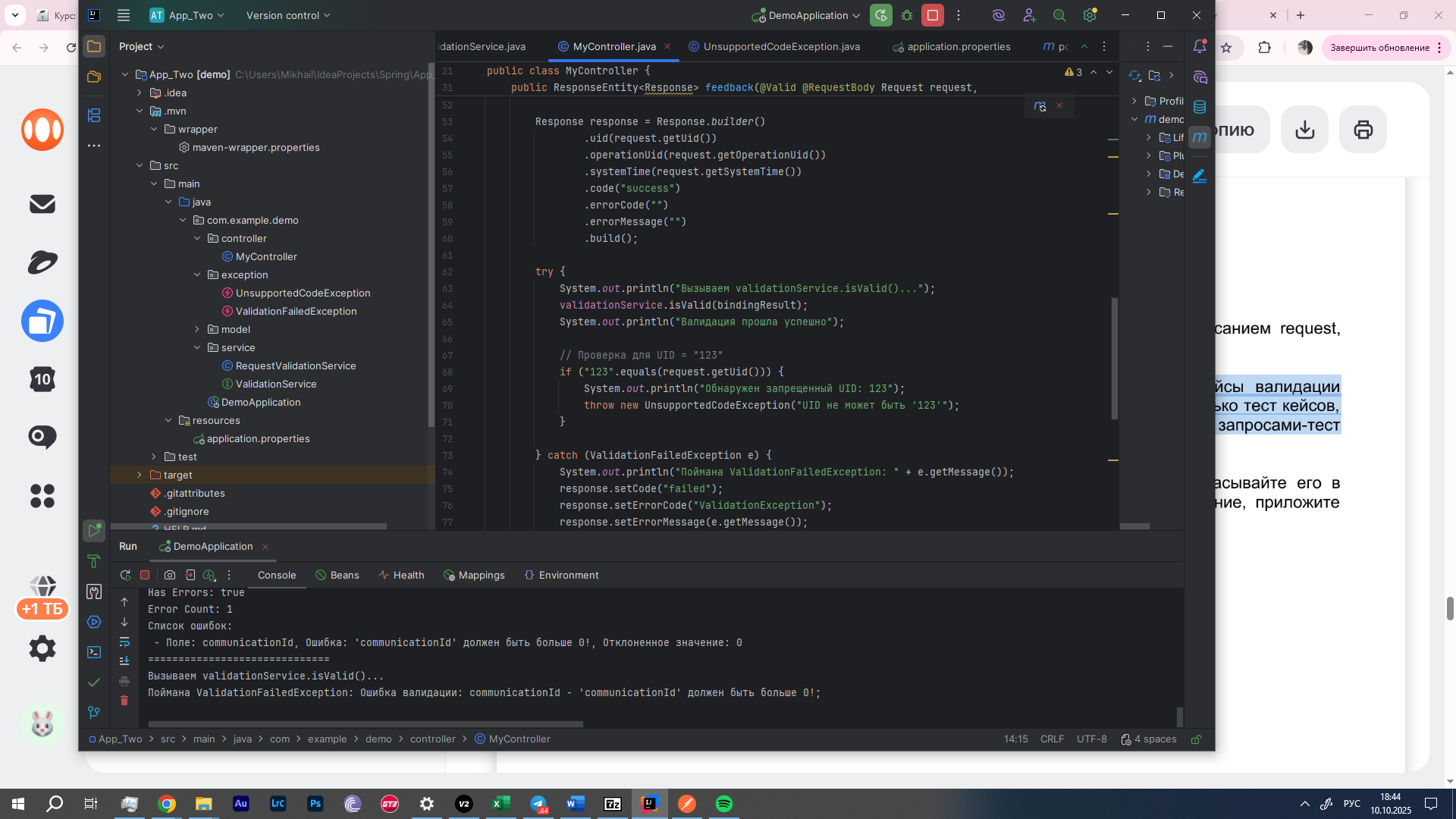


Рисунок 16 – Новая часть кода класса MyController

Протестируем снова нашу программу через Postman. Для проверки валидации, создаем еще один тест с UID = 123. Запрос должен быть с ошибкой. Результат теста показан на рисунке 17.

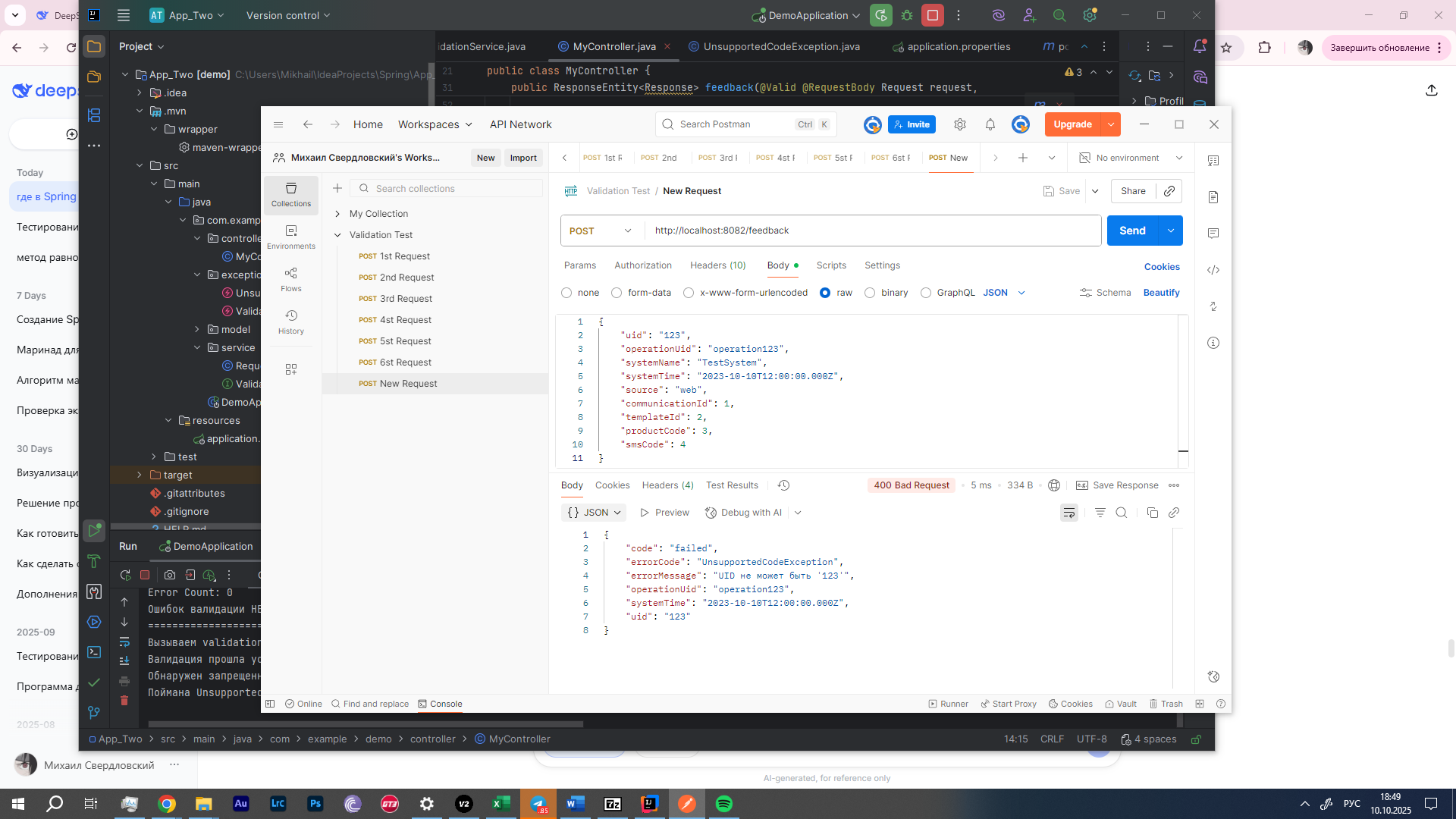


Рисунок 17 – Кейс теста с UID=123

**Задание 3. Ответы на вопросы.**

1 Чем отличаются GET запрос от POST запроса?

Для GET запроса: для получения данных – читается информация с сервера, параметры передаются в адресной строке, кешируются, длинна ограничена. Для POST изменяются данные на сервере, данные не видны в URL, каждый запрос уникален, возможна передача большого объема данных,

2 Какое содержимое может передаваться POST запросом?

Текстовые форматы, документы, изображения.

3 Чем отличается XML файл от JSON?

JSON легче и быстрее, его проще читать, лучше для веб-разработок. XLS имеет строгую структуру, в нем больше возможностей и поддерживаются метаданные.

4 Зачем нужны кастомные исключения?

Для тестирования программы или можно выделить специфические ошибки.

5 Что означает аннотация @Service?

Отмечает класс как сервисный компонент. Spring автоматически находит и определяет этот класс.

6 Что означает аннотация @RestController?

Используется для создания REST API. Автоматически отмечает класс как контроллер и преобразует возвращаемые объекты в JSON или XML.

**Вывод**

В результате лабораторной работы был разработан простой REST сервис.