|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**ОТЧЕТ ПО ПРАКТИЧЕСКИМ РАБОТАМ №1-8**

**по дисциплине**

«**Разработка клиент-серверных приложений**»

|  |  |
| --- | --- |
| Выполнил студент группы ИКБО-20-21 | Заводченков В.М. |
| Принял ассистент | Зарипов Е.А. |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2024 г.

ОГЛАВЛЕНИЕ

[ПРАКТИЧЕСКАЯ РАБОТА №1 3](#_Toc184853784)

[ПРАКТИЧЕСКАЯ РАБОТА №2 8](#_Toc184853785)

[ПРАКТИЧЕСКАЯ РАБОТА №3 15](#_Toc184853786)

[ПРАКТИЧЕСКАЯ РАБОТА №4 27](#_Toc184853787)

[ПРАКТИЧЕСКАЯ РАБОТА №5 33](#_Toc184853788)

[ПРАКТИЧЕСКАЯ РАБОТА №6 39](#_Toc184853789)

[ПРАКТИЧЕСКАЯ РАБОТА №7 45](#_Toc184853790)

[ПРАКТИЧЕСКАЯ РАБОТА №8 50](#_Toc184853791)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 51](#_Toc184853792)

# **ПРАКТИЧЕСКАЯ РАБОТА №1**

Данная практическая работа в полном объеме загружена на GitHub и доступна для просмотра по данной ссылке - <https://github.com/SlasherSDCaT/MireaSem7/tree/main/rksp/pract_1_4>

**Задание на практическую работу**

**Задание 1**

Дан массив из 10000 элементов. Необходимо написать несколько реализаций некоторой функции F в зависимости от варианта. Функция должна быть реализована следующими способами:

1. Последовательно

2. С использованием многопоточности (Thread, Future, и т. д.)

3. С использованием ForkJoin.

После каждой операции с элементом массива (сравнение, сложение) добавить задержку в 1 мс при помощи Thread.sleep(1);

Провести сравнительный анализ затрат по времени и памяти при запуске каждого из вариантов реализации.

Варианты функций (выбор варианта осуществляется по формуле «Номер в списке группы % 3»)

1. Поиск суммы элементов массива.

2. Поиск максимального элемента в массиве.

3. Поиск минимального элемента в массиве.

**Выполнение задания**

Код представлен на рисунке 1.1. Выполнение представлено на рисунке 1.2.

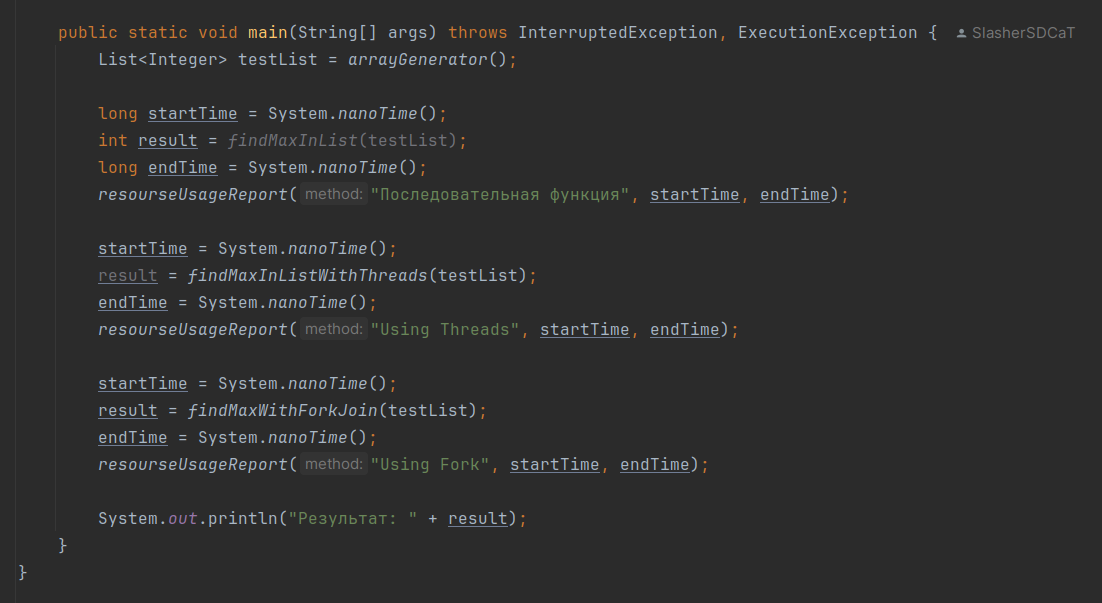


Рисунок 1.1 – Задание 1

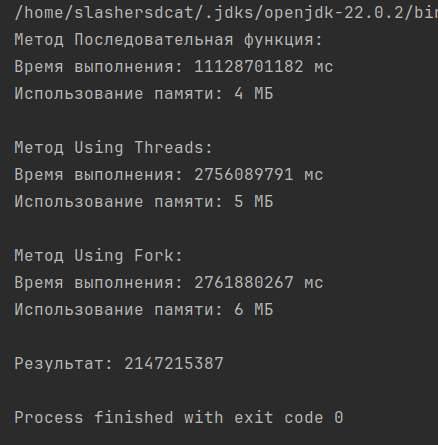


Рисунок 1.2 – Результат задания 1

**Задание 2** Программа запрашивает у пользователя на вход число. Программа имитирует обработку запроса пользователя в виде задержки от 1 до 5 секунд выводит результат: число, возведенное в квадрат. В момент выполнения запроса пользователь имеет возможность отправить новый запрос. Реализовать с использованием Future.

**Выполнение задания**

Задание представлено 1.3. Выполнение представлено на рисунке 1.4.

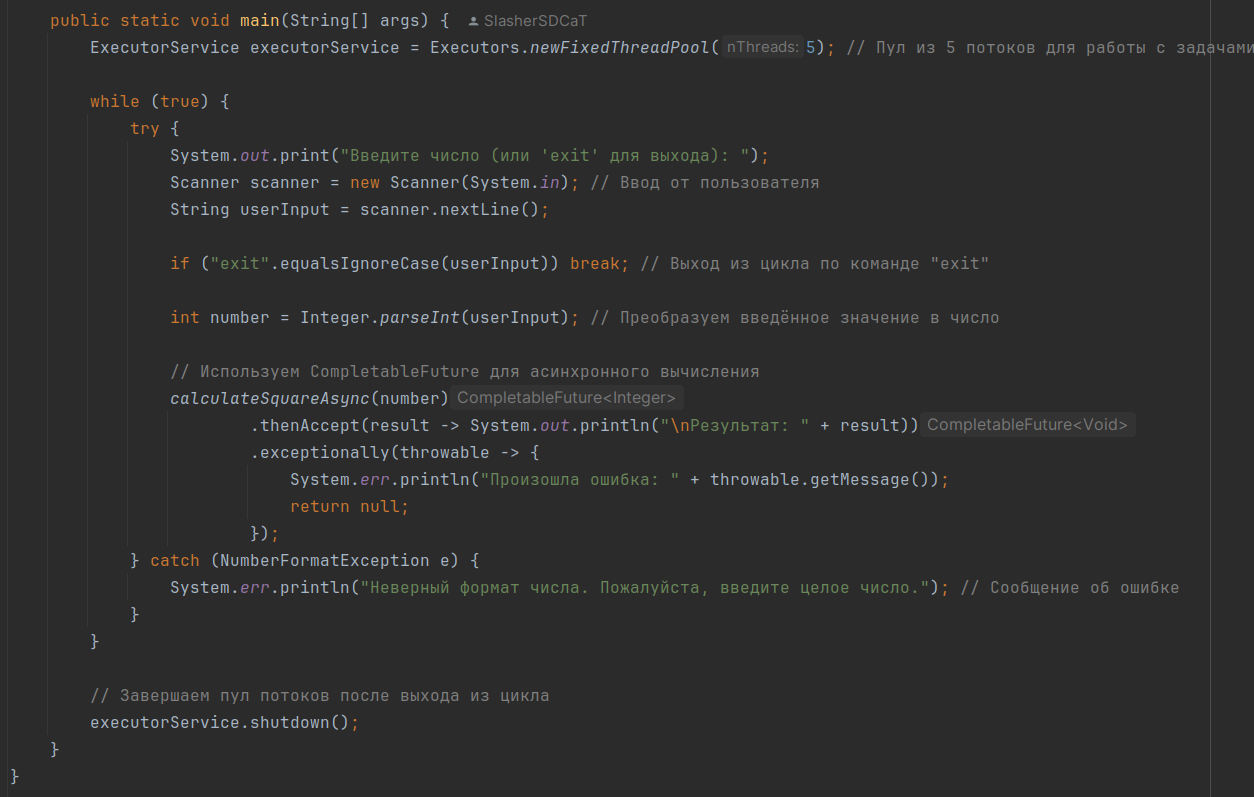


Рисунок 1.3 – Задание 2

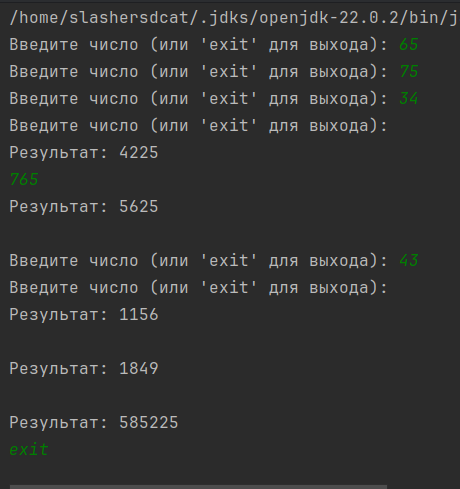


Рисунок 1.4 – Выполнение задания 2

**Задание 3**

Реализовать следующую многопоточную систему. Файл. Имеет следующие характеристики:

1. Тип файла (например XML, JSON, XLS)

2. Размер файла — целочисленное значение от 10 до 100

Генератор файлов - генерирует файлы с задержкой от 100 до 1000 мс.

Очередь — получает файлы из генератора. Вместимость очереди — 5

Обработчик файлов — получает файл из очереди. Каждый обработчик имеет параметр — тип файла, который он может обработать. Время обработки файла: «Размер файла\*7мс»

**Выполнение задания**

Задание представлено на рисунке 1.5. Выполнение представлено на рисунке 1.6.

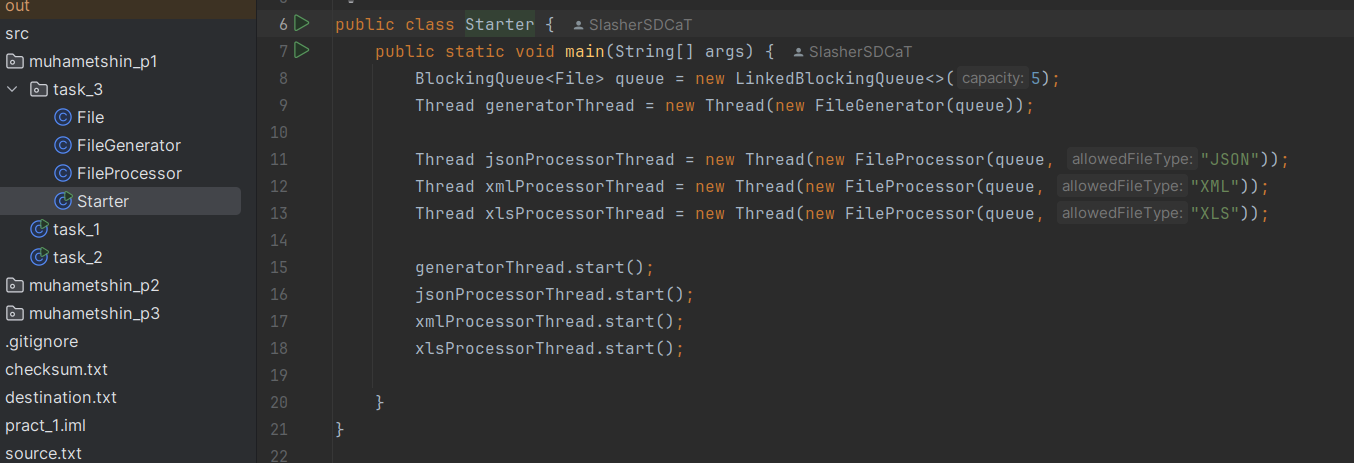


Рисунок 1.5 – Задание 3

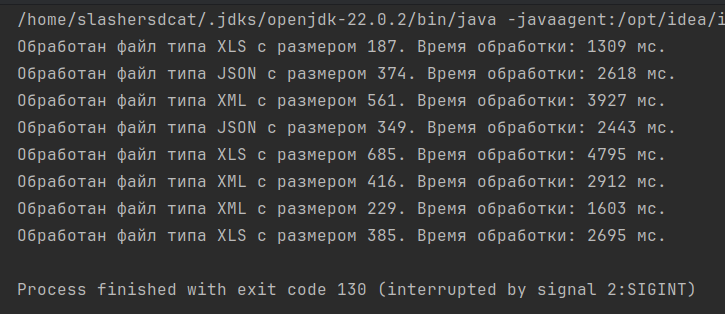


Рисунок 1.6 – Выполнение задание 3

# **ПРАКТИЧЕСКАЯ РАБОТА №2**

Данная практическая работа в полном объеме загружена на GitHub и доступна для просмотра по данной ссылке - <https://github.com/SlasherSDCaT/MireaSem7/tree/main/rksp/pract_1_4>

**Задание на практическую работу**

**Задание 1**

Создать файл формата .txt, содержащий несколько строк текста. С помощью пакета java.nio нужно прочитать содержимое файла и вывести данные в стандартный поток вывода.

**Выполнение задания**

Код и выполнение представлены на рисунках 2.1-2.2.

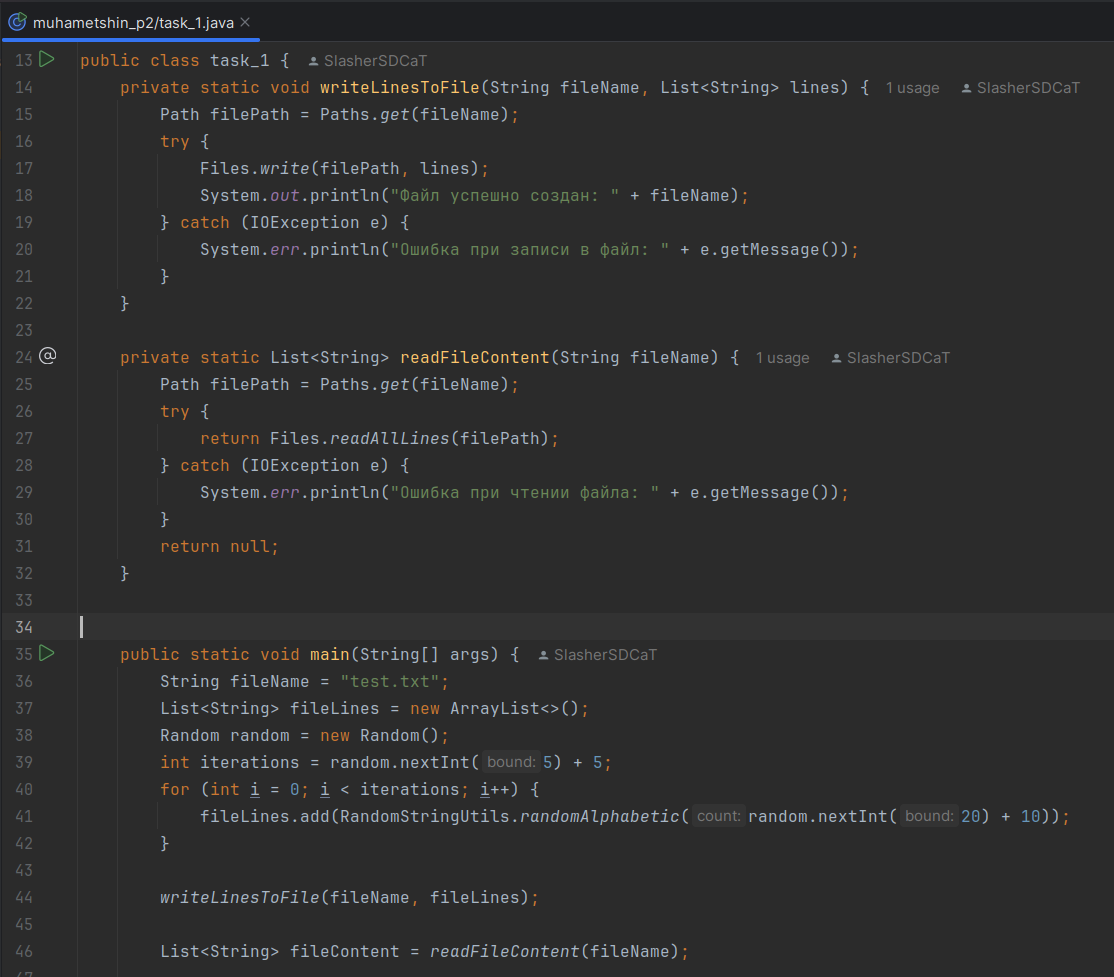


Рисунок 2.1 – Задание 1

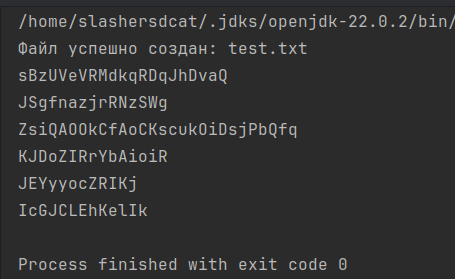


Рисунок 2.2 – Задание 1

**Задание 2**

Реализовать копирование файла размером 100 Мб 4 методами:

1) FileInputStream/FileOutputStream

2) FileChannel

3) Apache Commons IO

4) Files class

Замерить затраты по времени и памяти и провести сравнительный анализ.

**Выполнение задания**

Код и выполнение представлены на рисунках 2.3-2.8.

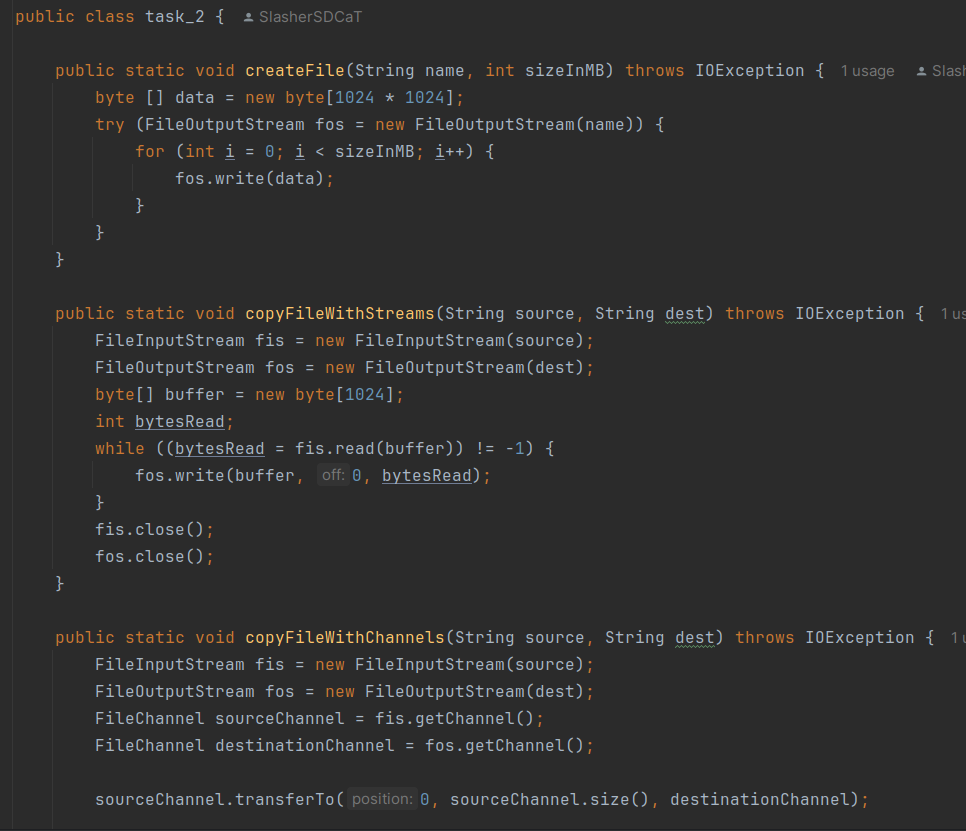


Рисунок 2.3 – Задание 2

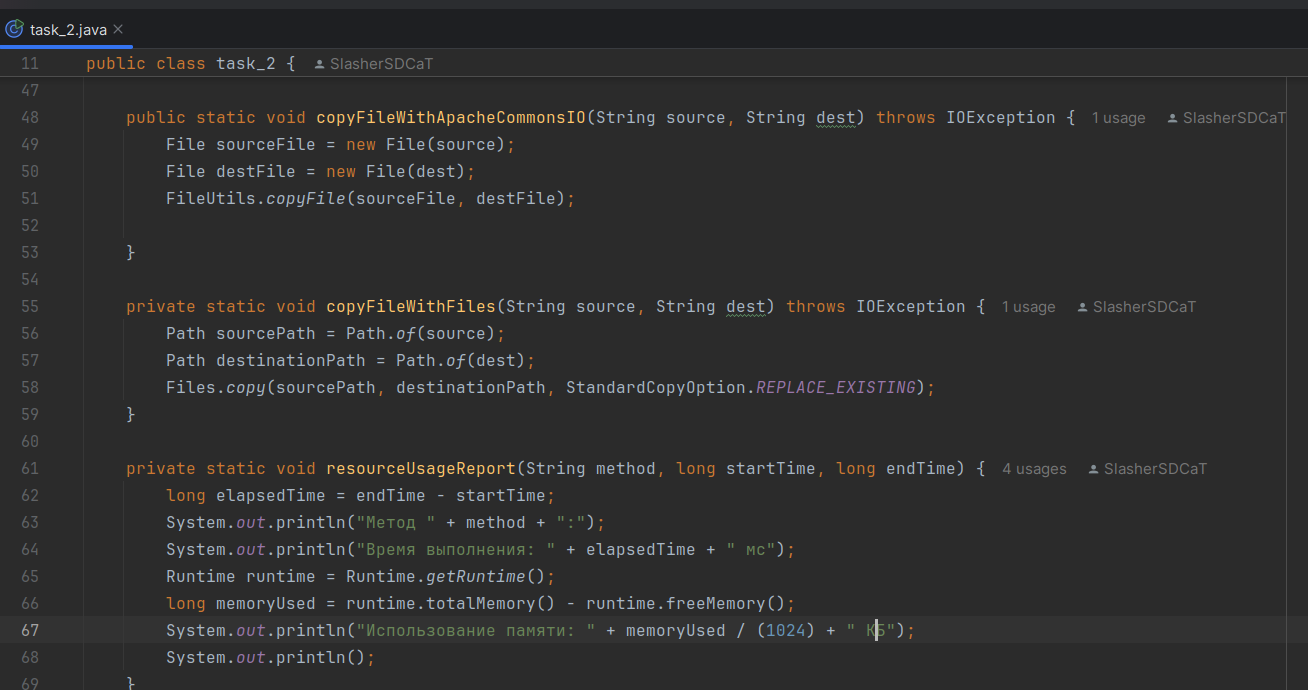


Рисунок 2.4 – Задание 2

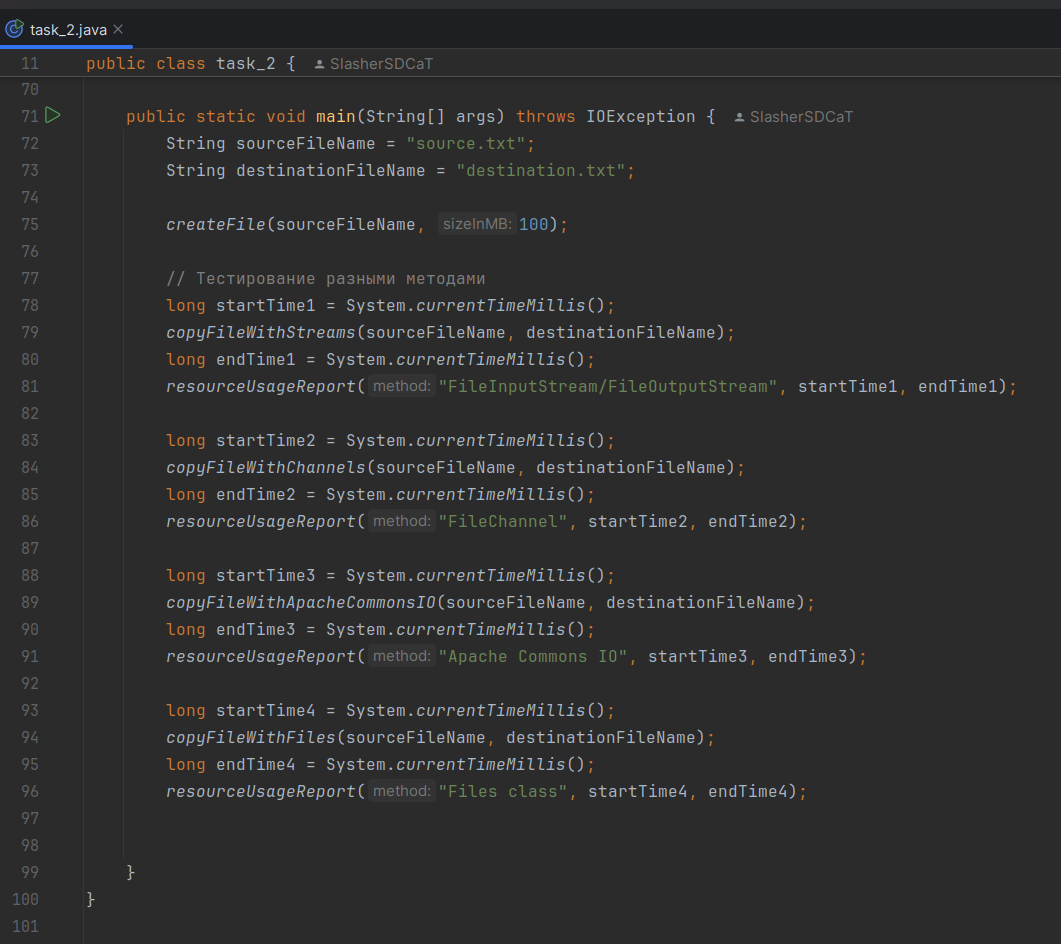


Рисунок 2.5 – Задание 2

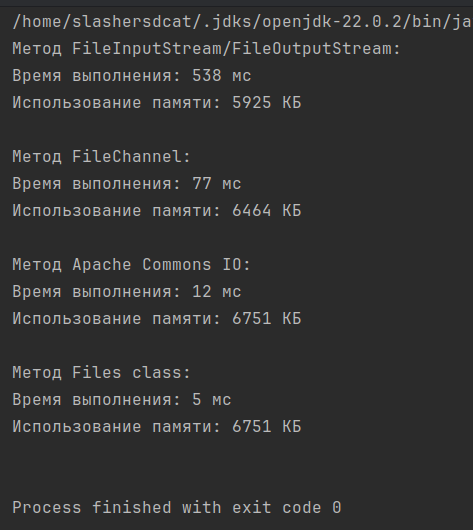


Рисунок 2.6 – Выполнение задания 2

**Задание 3**

Реализовать функцию нахождения 16-битной контрольной суммы файла с использованием бинарных операций и ByteBuffer.

**Выполнение задания**

Код и выполнение представлены на рисунках 2.7-2.8.

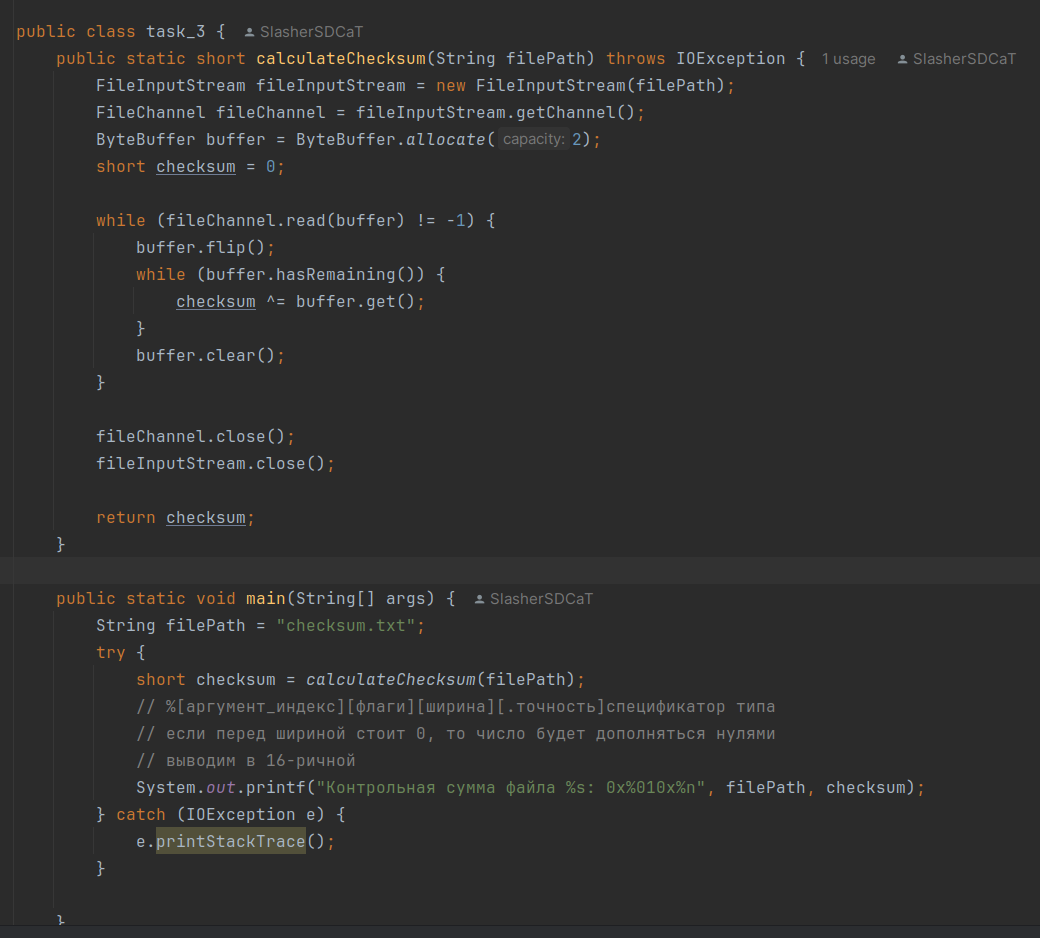


Рисунок 2.7 – Задание 3

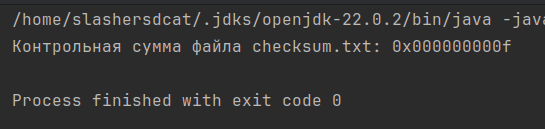


Рисунок 2.8 – Выполнение задания 3

**Задание 4**

При помощи WatchService реализовать наблюдение за каталогом:

1) При создании нового файла в этом каталоге вывести его название;

2) При изменении файла вывести список изменений(добавленных и

удаленных строк);

3) При удалении файла вывести его размер и контрольную сумму.

**Выполнение задания**

Код и выполнение представлены на рисунках 2.9-2.10.

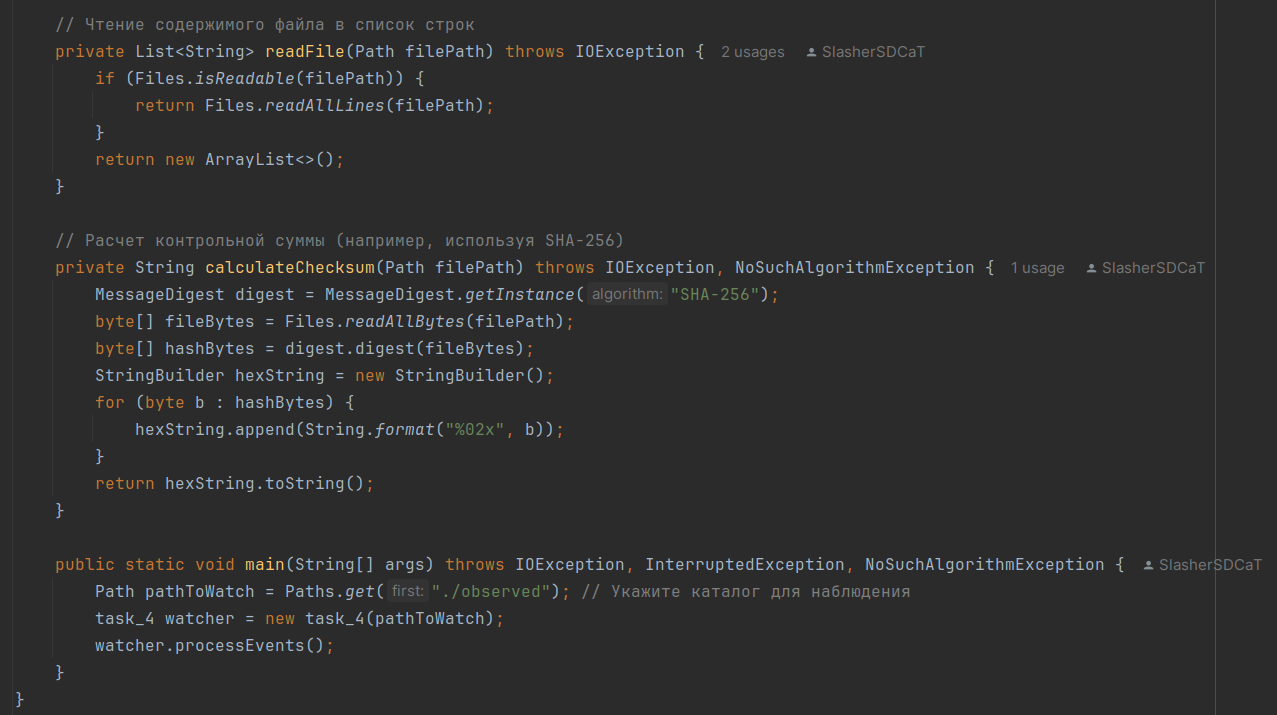


Рисунок 2.9 – Задание 4

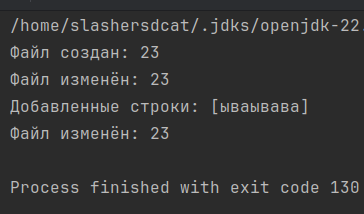


Рисунок 2.10 – Выполнение задания 4

# **ПРАКТИЧЕСКАЯ РАБОТА №3**

Данная практическая работа в полном объеме загружена на GitHub и доступна для просмотра по данной ссылке - <https://github.com/SlasherSDCaT/MireaSem7/tree/main/rksp/pract_1_4>

**Задание на практическую работу**

**Задание 1**

Реализовать следующую систему:

Датчик температуры. Каждую секунду публикует значение температуры

(случайное значение от 15 до 30). Датчик CO2. Каждую секунду публикует значение содержания CO2 в воздухе. (Случайное значение от 30 до 100). Сигнализация. Получает значения от датчиков. Если один из показателей превышает норму, выводит предупреждение об этом. Если норму превышаю оба показателя выводит сообщение «ALARM!!!». Норма показателей: Температура — 25 CO2 — 70

Обязательно использование классов Observer и Observable из библиотеки RxJava.

**Выполнение задания**

Код и выполнение представлены на рисунках 3.1-3.3.

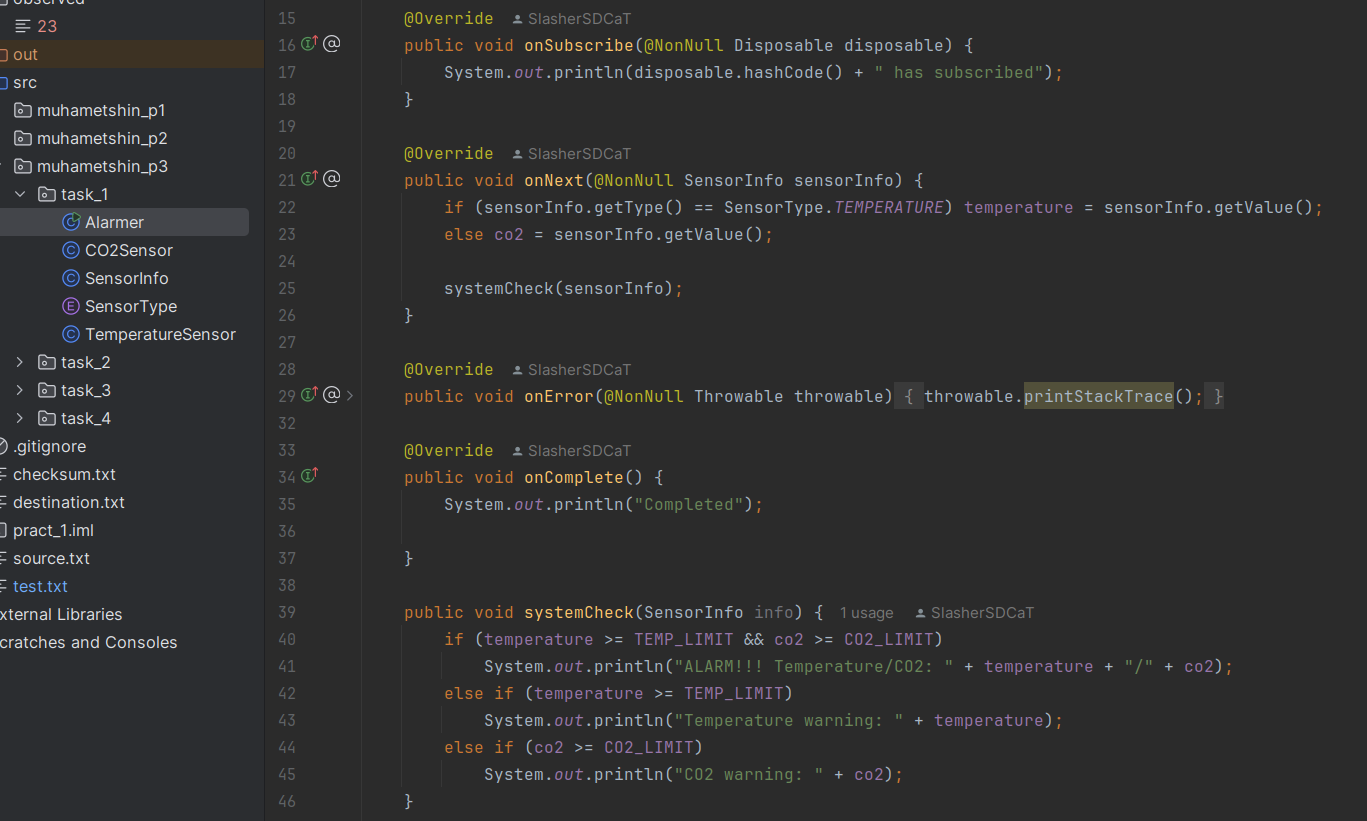


Рисунок 3.1 – Задание 1

Рисунок 3.2 – Задание 1

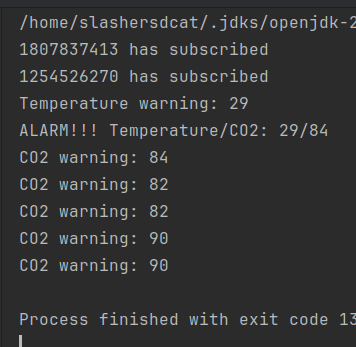


Рисунок 3.3 – Задание 1

**Задание 2**

2.1.1 Преобразовать поток из 1000 случайных чисел от 0 до 1000 в поток, содержащий квадраты данных чисел.

2.2.1. Даны два потока по 1000 элементов: первый содержит случайную букву, второй — случайную цифру. Сформировать поток, каждый элемент которого объединяет элементы из обоих потоков. Например, при входных потоках (A, B, C) и (1, 2, 3) выходной поток — (A1, B2, B3).

2.3.1. Дан поток из 10 случайных чисел. Сформировать поток, содержащий все числа, кроме первых трех.

**Выполнение задания**

Код и выполнение представлены на рисунках 3.4-3.8.

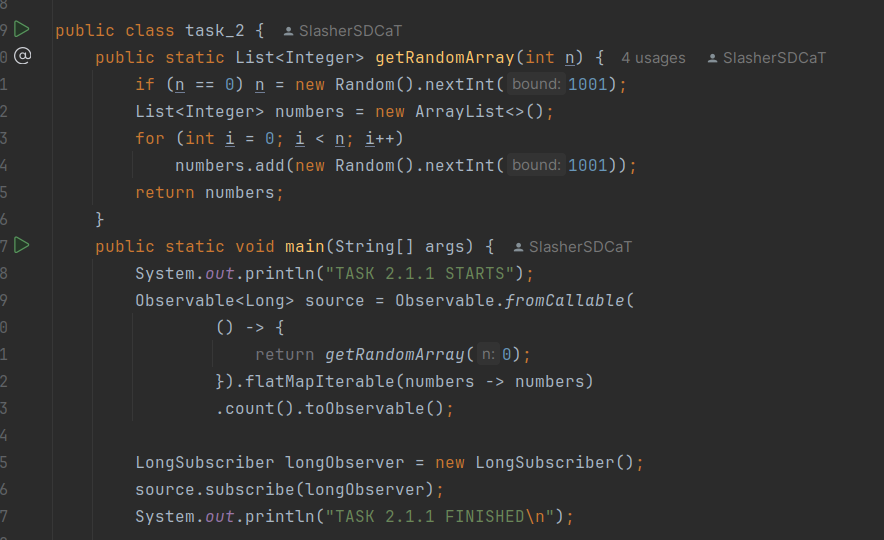


Рисунок 3.4 – Задание 2.1.1

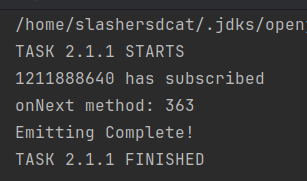


Рисунок 3.5 – Задание 2.1.1

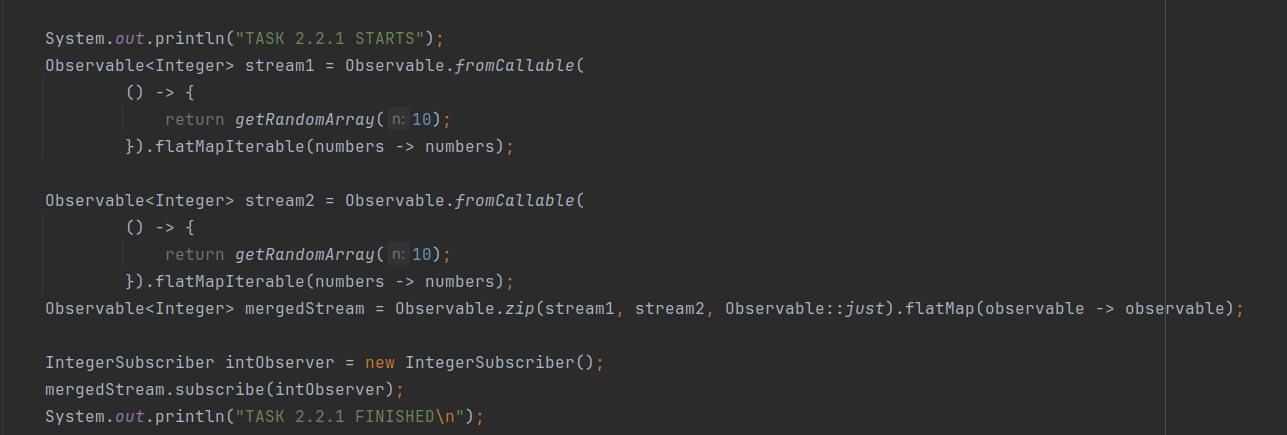


Рисунок 3.6 – Задание 2.2.1

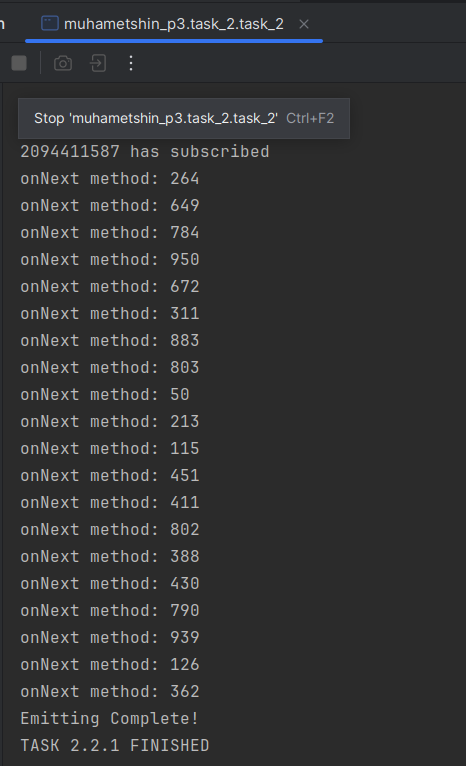


Рисунок 3.7 – Задание 2.2.1

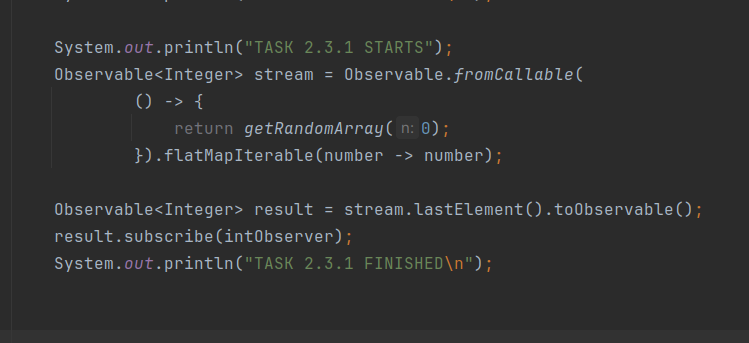


Рисунок 3.8 – Задание 2.3.1

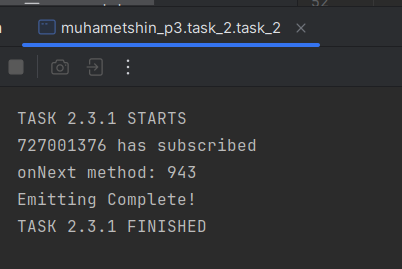


Рисунок 3.9 – Задание 2.3.1

**Задание 3**

Реализовать класс UserFriend. Поля — int userId, friendId. Заполнить массив объектов UserFriend случайными данными.

Реализовать функцию: Observable<UserFriend> getFriends(int userId), возвращающую поток объектов UserFriend, по заданному userId. (Для формирования потока из массива возможно использование функции Observable.fromArray(T[] arr)).

Дан массив из случайных userId. Сформировать поток из этого массива. Преобразовать данный поток в поток объектов UserFriend. Обязательно получение UserFriend через функцию getFriends.

**Выполнение задания**

Код и выполнение представлены на рисунках 3.10-3.11.

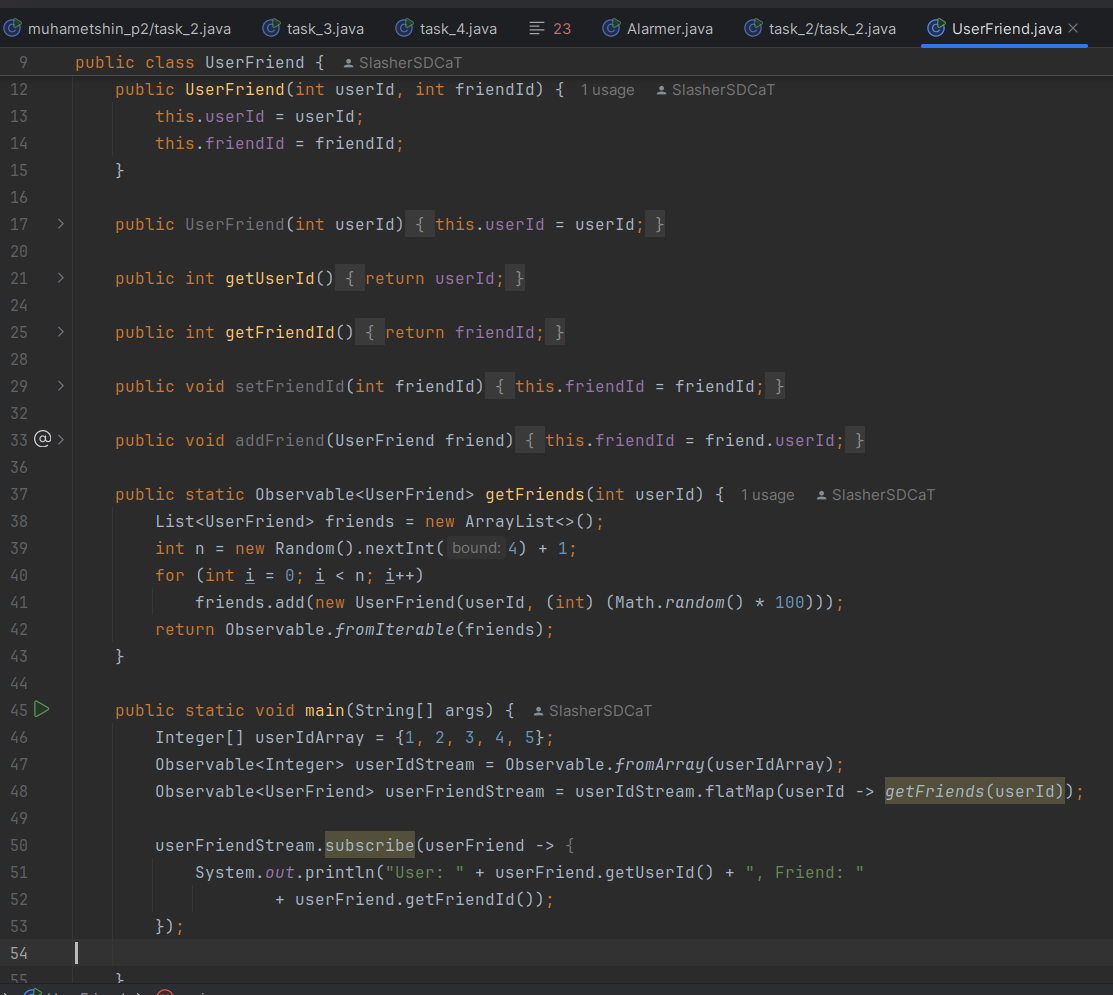


Рисунок 3.10 – Задание 3

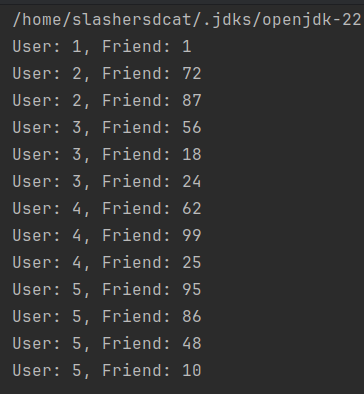


Рисунок 3.11 – Задание 3

**Задание 4**

Реализовать следующую систему.

Файл. Имеет следующие характеристики:

1. Тип файла (например XML, JSON, XLS)
2. Размер файла — целочисленное значение от 10 до 100.

Генератор файлов — генерирует файлы с задержкой от 100 до 1000 мс.

Очередь — получает файлы из генератора. Вместимость очереди — 5 файлов.

Обработчик файлов — получает файл из очереди. Каждый обработчик имеет параметр — тип файла, который он может обработать. Время обработки файла: «Размер файла\*7мс». Система должна быть реализована при помощи инструментов RxJava.

**Выполнение задания**

Код и выполнение представлены на рисунках 3.12-3.17.

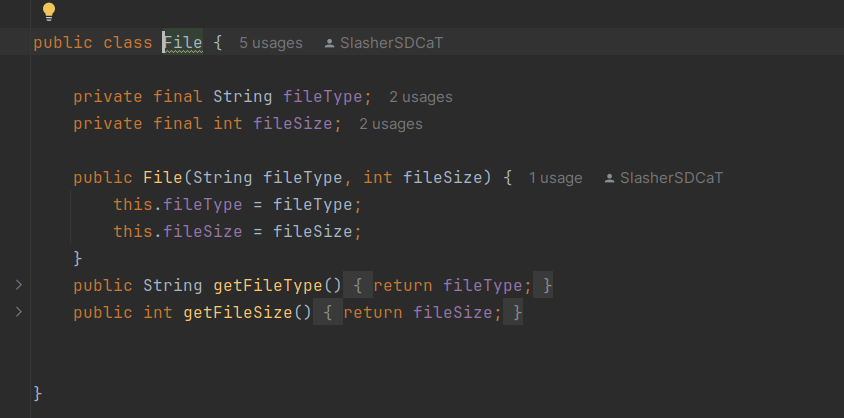


Рисунок 3.12 – Задание 4

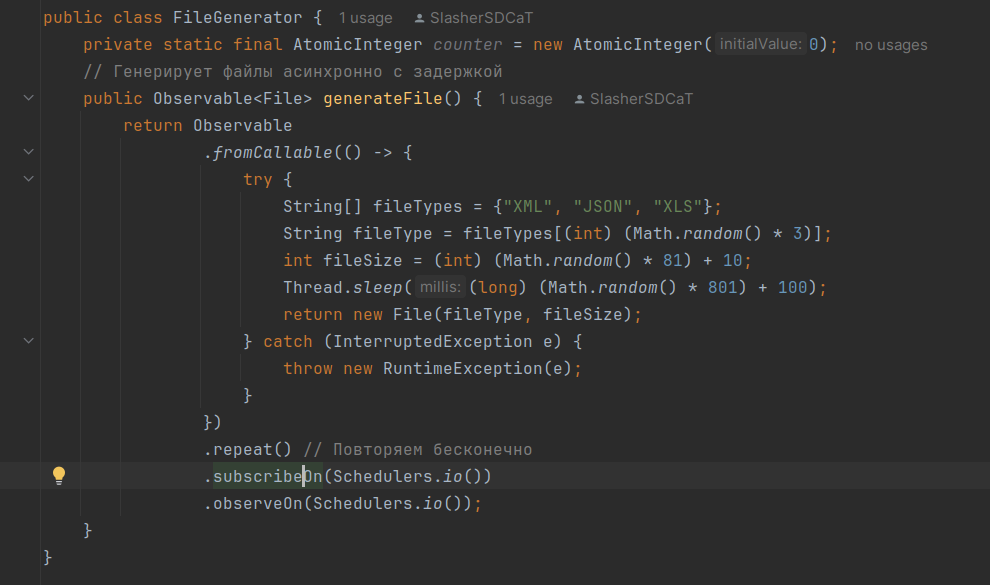


Рисунок 3.13 – Задание 4

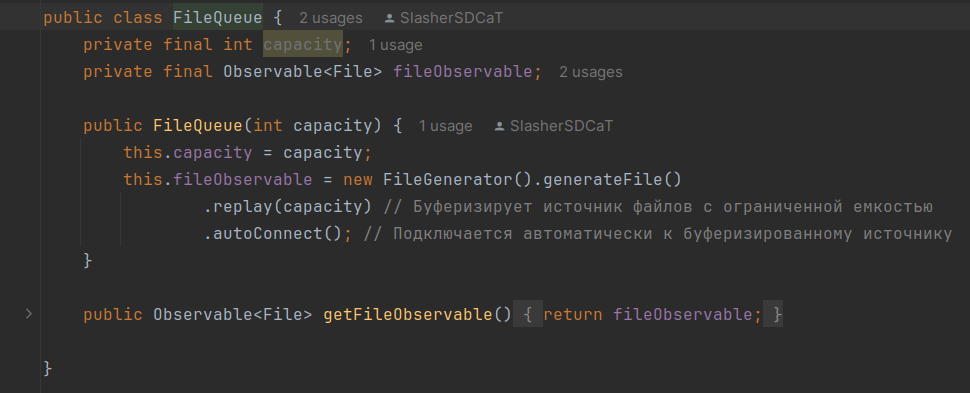


Рисунок 3.14 – Задание 4

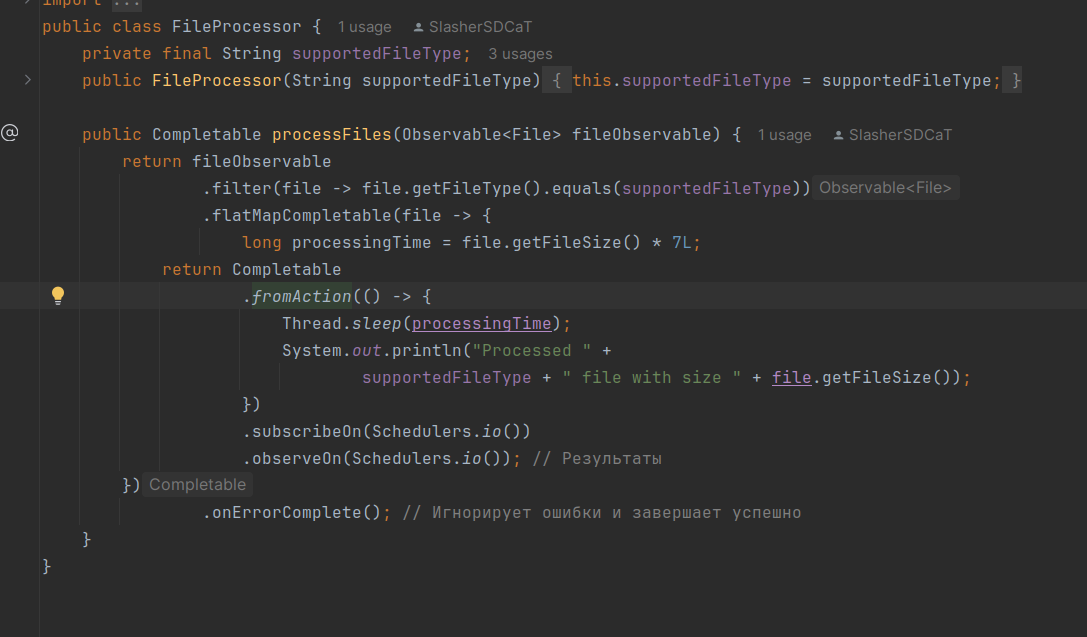


Рисунок 3.15 – Задание 4

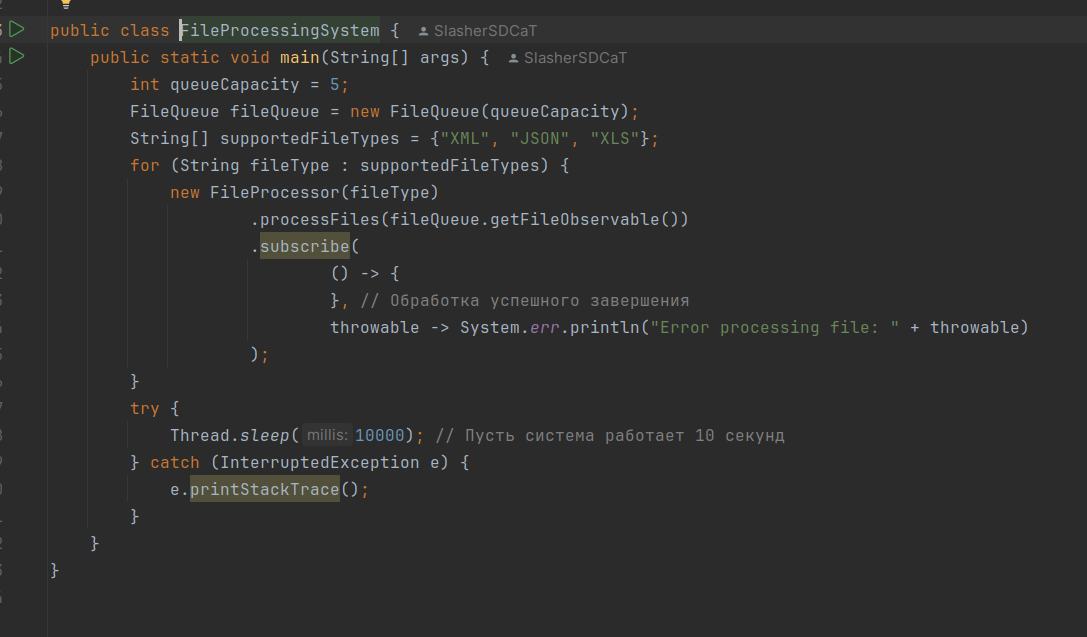


Рисунок 3.16 – Задание 4

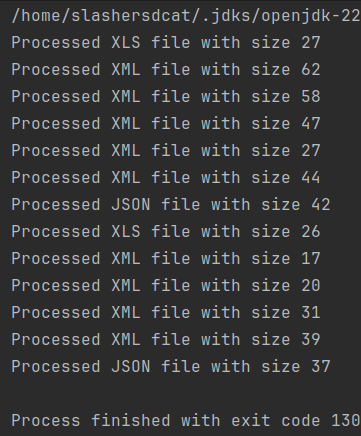


Рисунок 3.17 – Задание 4

# **ПРАКТИЧЕСКАЯ РАБОТА №4**

Данная практическая работа в полном объеме загружена на GitHub и доступна для просмотра по данной ссылке - <https://github.com/SlasherSDCaT/MireaSem7/tree/main/rksp/4>

**Задание на практическую работу**

**Задание**

Разработать клиент-серверную систему с использованием протокола RSocket.

Предметная область разрабатываемой системы выбирается студентом самостоятельно, но должна быть уникальна в рамках учебной группы.

Система должна удовлетворять следующим требованиям:

* Должны быть продемонстрированы все 4 типа взаимодействия в рамках протокола RSocket:
  + Request-Response;
  + Request-Stream;
  + Channel;
  + Fire-and-Forget.
* Должна быть продемонстрирована работа с операторами преобразования потоков.
* Должно быть продемонстрировано взаимодействие с базой данных.
* Потоки должны закрываться после отработки бизнес-логики.
* Серверная часть приложения должна быть покрыта Unit-тестами.

**Выполнение задания**

Код представлен на рисунках 4.1-4.7

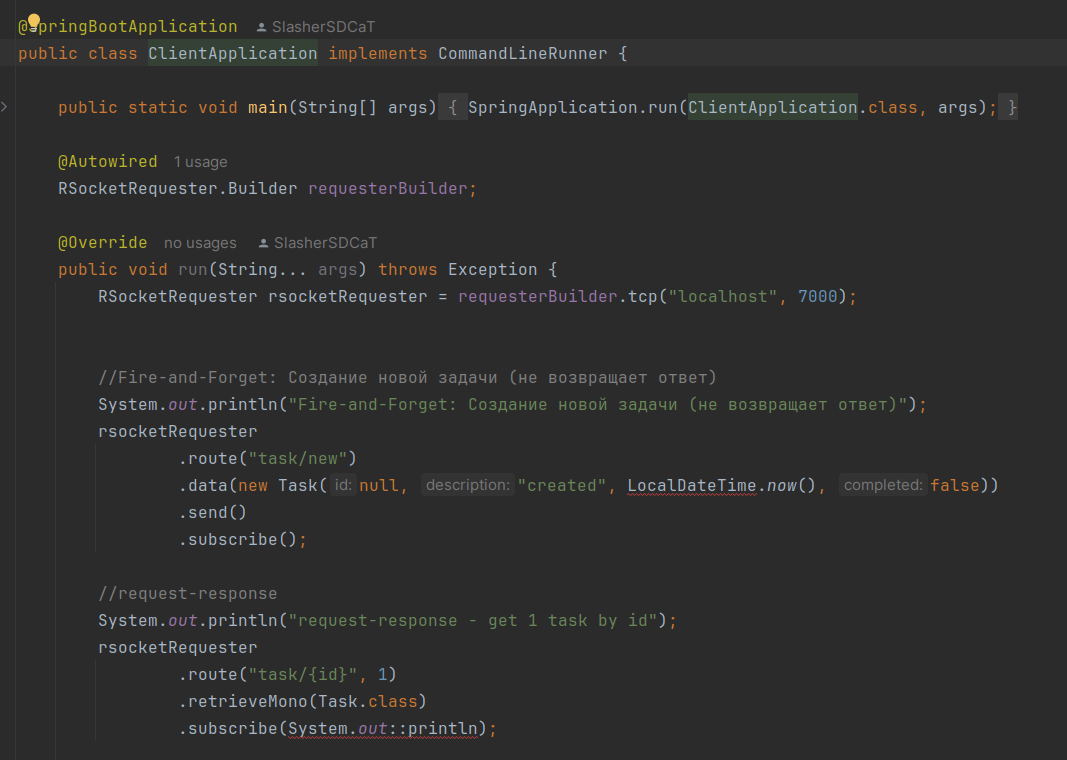


Рисунок 4.1 – Задание 1

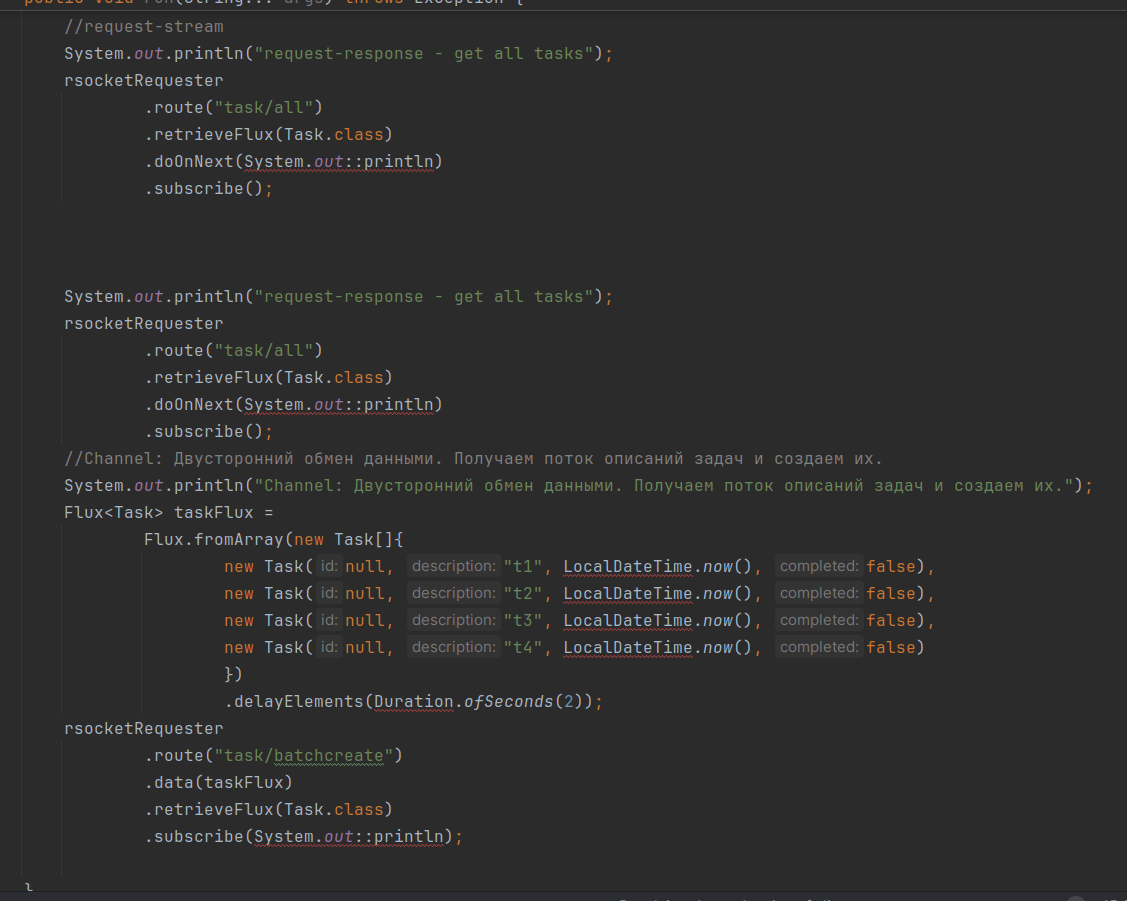


Рисунок 4.2 – Задание 1

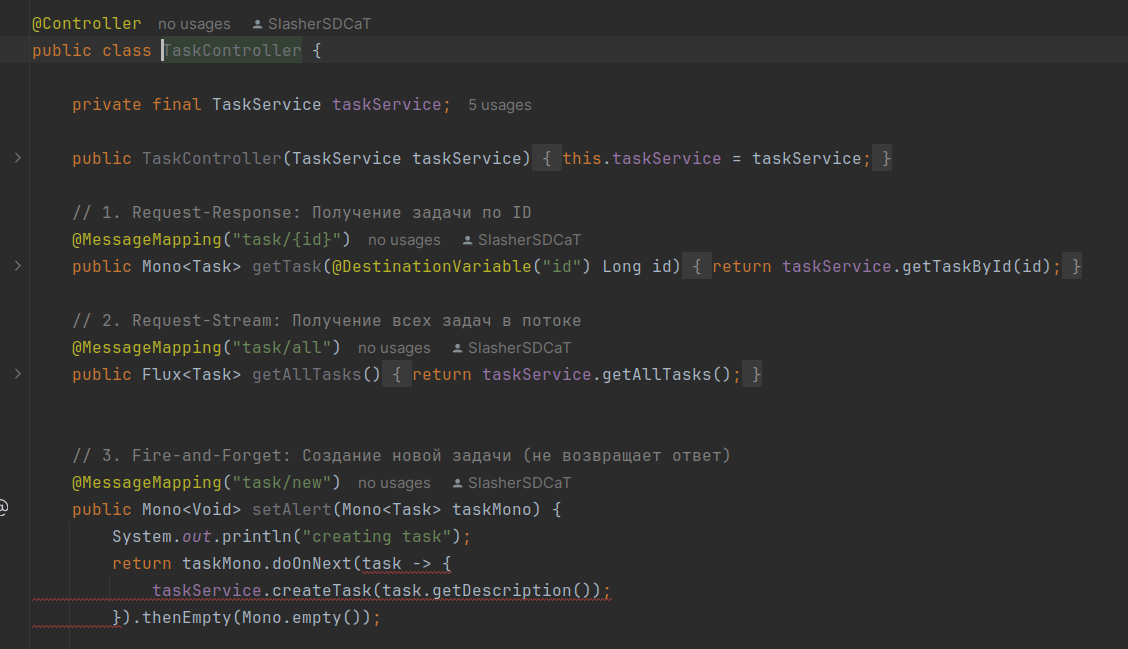


Рисунок 4.3 – Задание 1

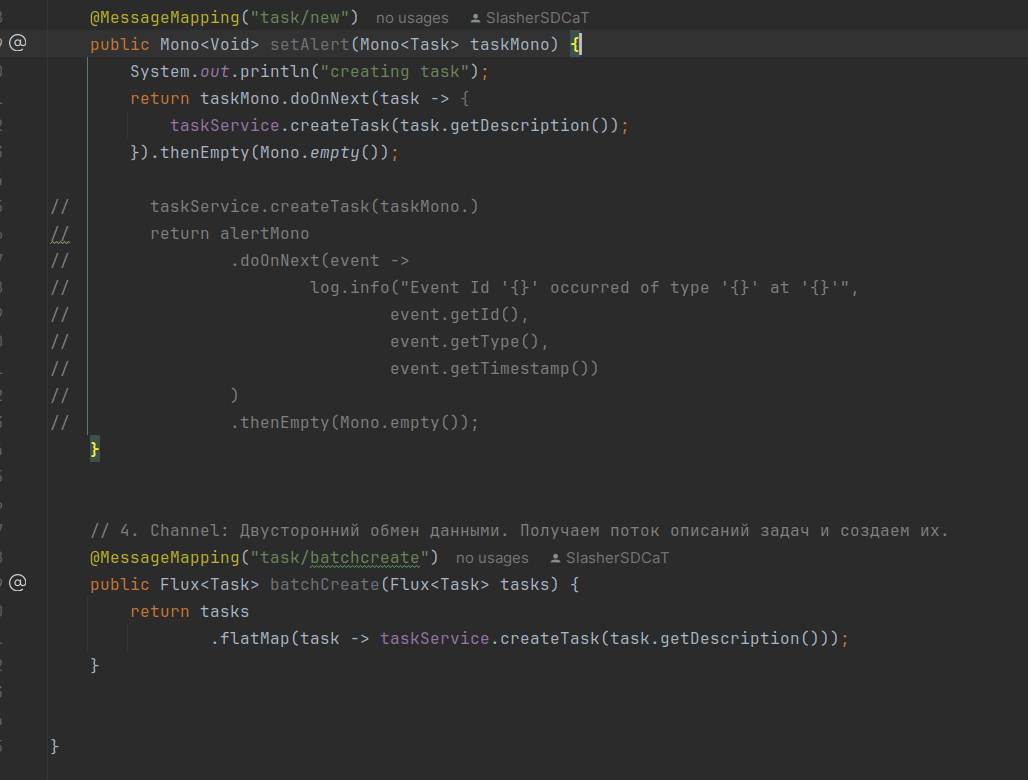


Рисунок 4.4 – Задание 1

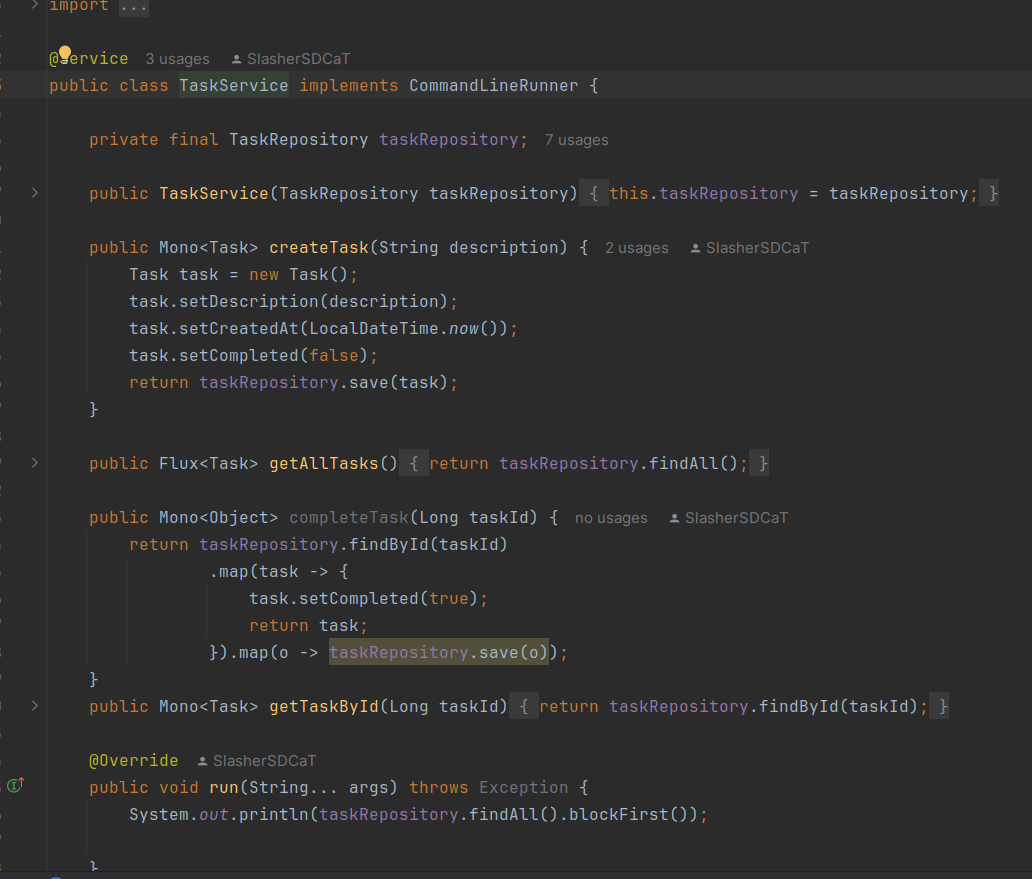


Рисунок 4.5 – Задание 1

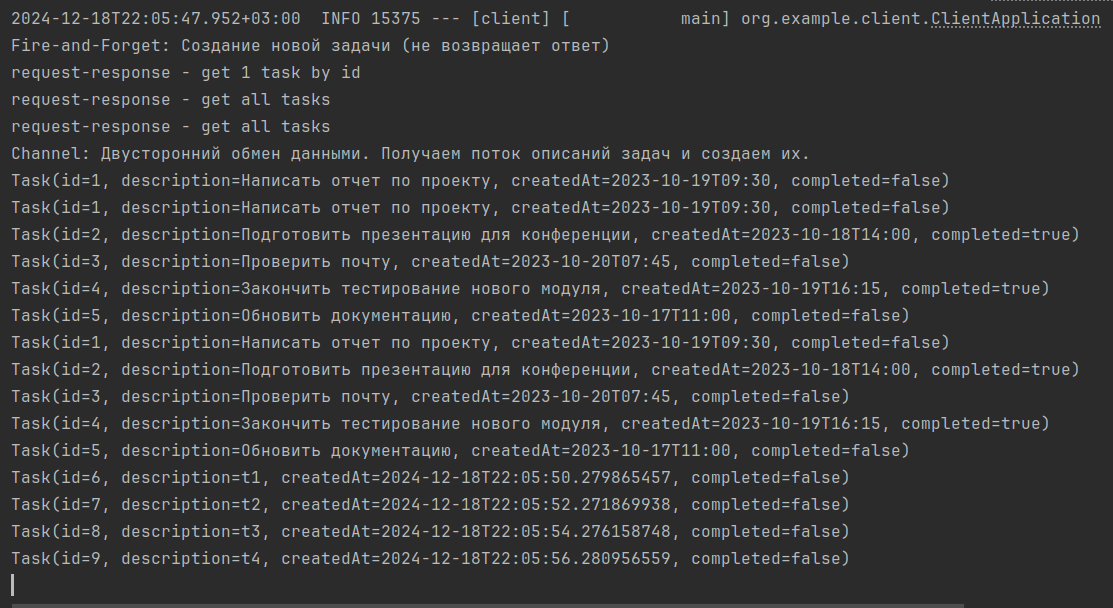


Рисунок 4.6 – Задание 1

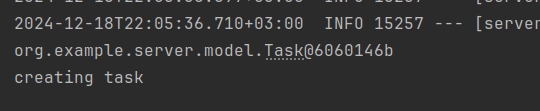


Рисунок 4.7 – Задание 1

# **ПРАКТИЧЕСКАЯ РАБОТА №5**

Данная практическая работа в полном объеме загружена на GitHub и доступна для просмотра по данной ссылке - <https://github.com/SlasherSDCaT/MireaSem7/tree/main/rksp/Practice5>

**Задание на практическую работу**

**Задание**

Разработать децентрализованную распределенную систему по согласованной теме с преподавателем с использованием Spring/Spring Boot.

Требуемый функционал системы: загрузка и выгрузка файлов различных форматов, также допустим функционал по индивидуальному варианту.

Для демонстрации системы необходимо развернуть как минимум 4 экземпляра приложения при помощи Docker и продемонстрировать работоспособность при отключении каждого из 4х экземпляров, а также при отказе любых двух экземпляров.

В случае невозможности реализации системы с использованием предложенного стека технологий необходимо обосновать это и предложить решение на собственном стеке.

**Выполнение задания**

Задание было выполнено в соответствии со всеми требованиями. Код представлен на рисунках 5.1-5.9

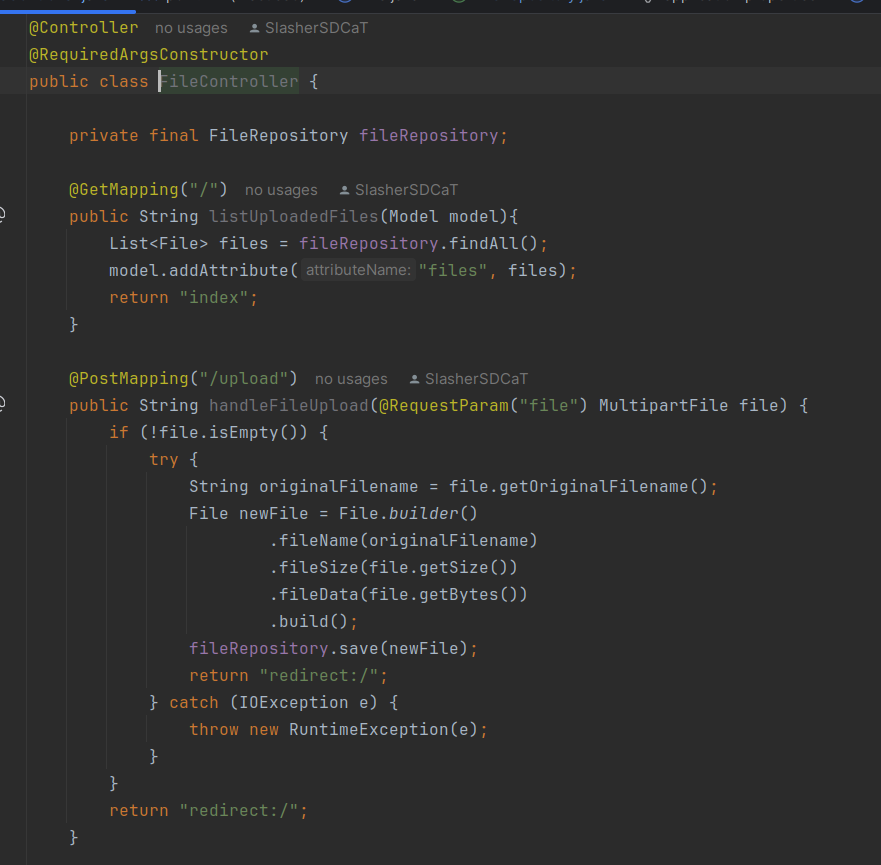


Рисунок 5.1 – Задание 1



Рисунок 5.2 – Задание 1

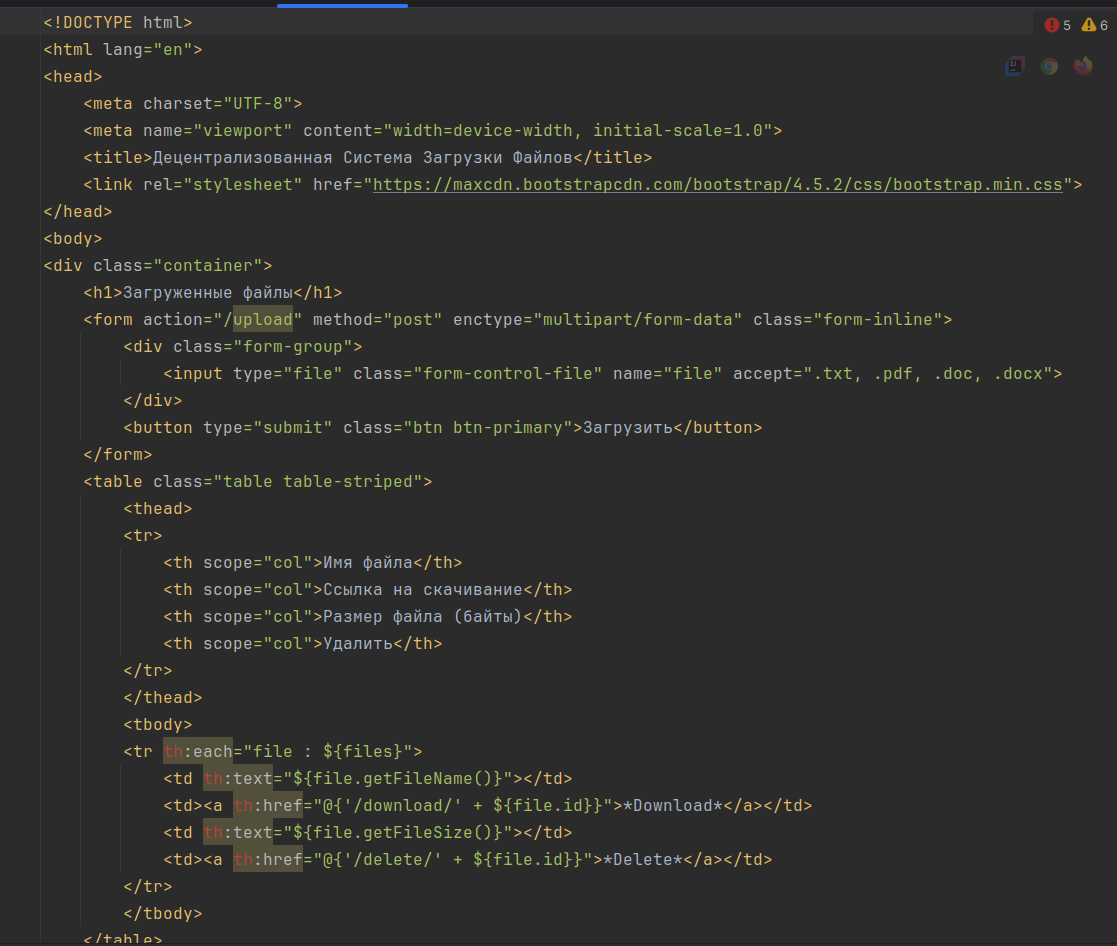


Рисунок 5.3 – Задание 1

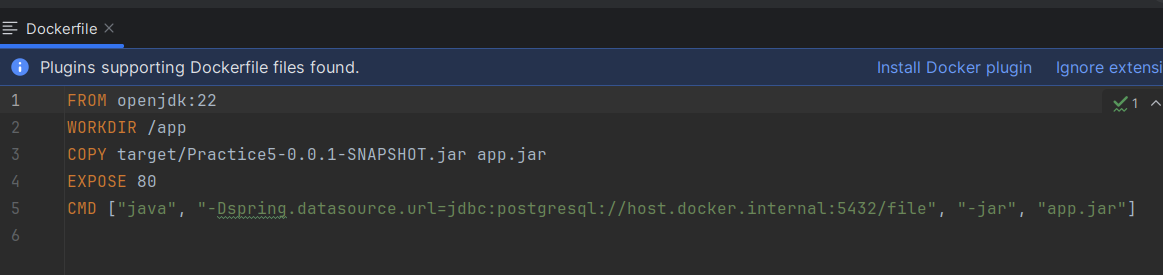


Рисунок 5.4 – Задание 1

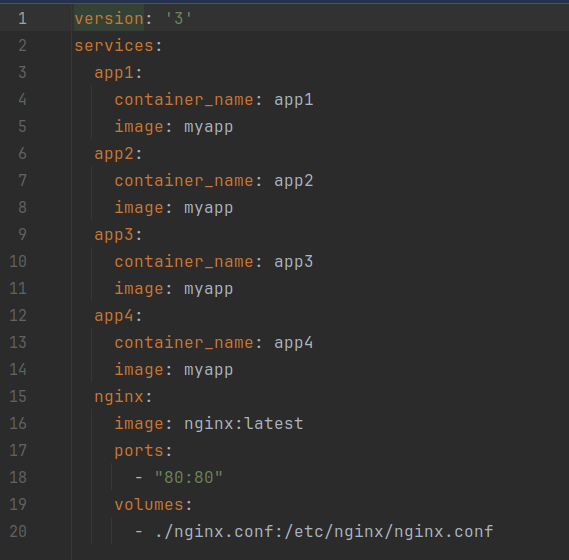


Рисунок 5.5 – Задание 1

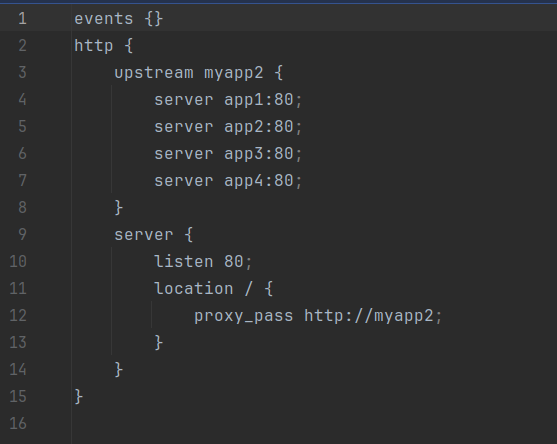


Рисунок 5.6 – Задание 1

Результат работы программы представлен на рисунках 5.10-5.

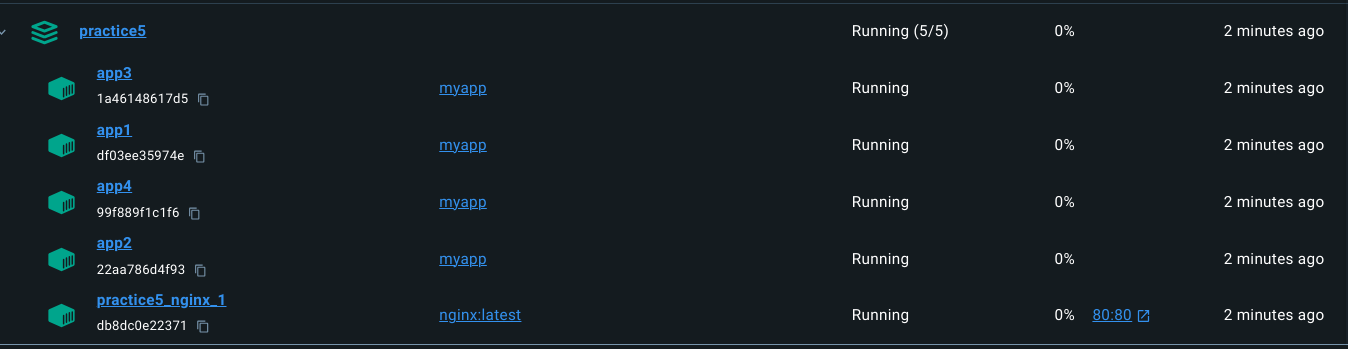


Рисунок 5.7 – Результат выполнения

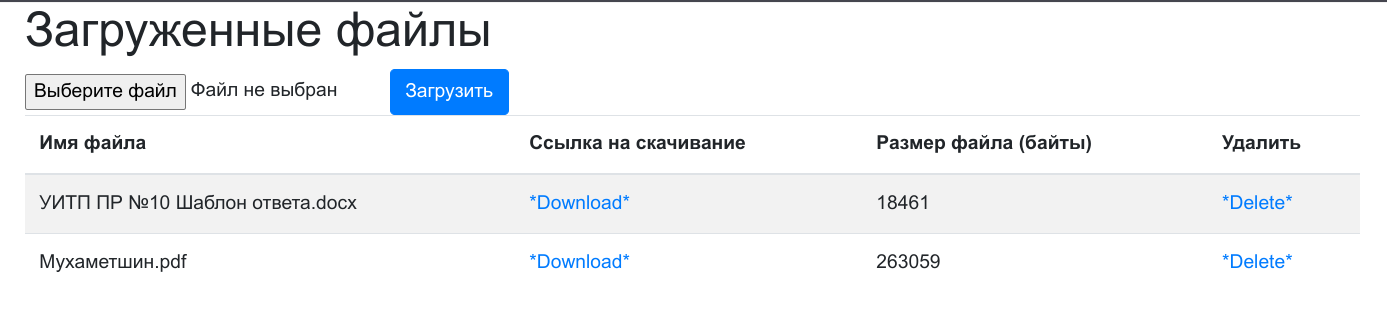


Рисунок 5.8 – Результат выполнения

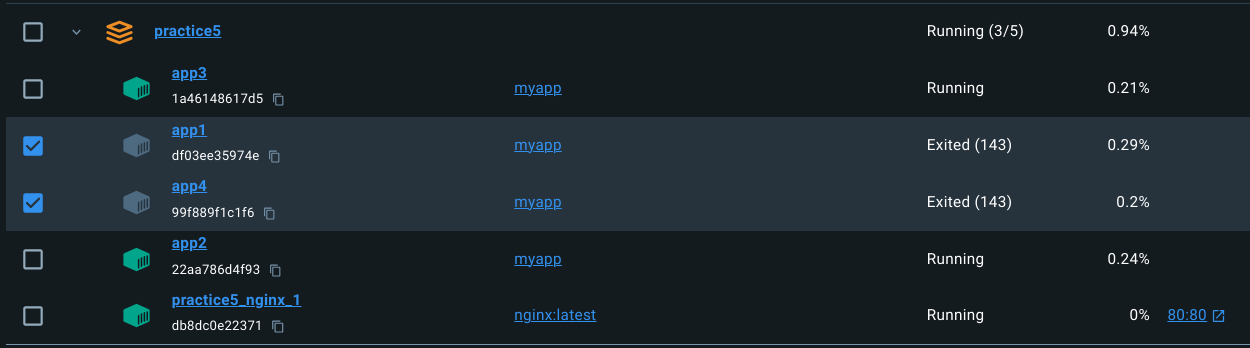


Рисунок 5.9 – Результат выполнения

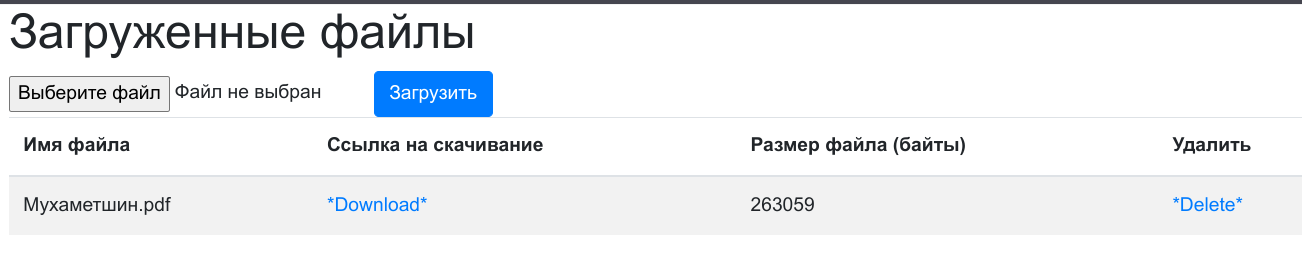


Рисунок 5.10 – Результат выполнения

# **ПРАКТИЧЕСКАЯ РАБОТА №6**

Данная практическая работа в полном объеме загружена на GitHub и доступна для просмотра по данной ссылке - <https://github.com/SlasherSDCaT/MireaSem7/tree/main/rksp/Practice6>

**Задание на практическую работу**

**Задание**

1. Написать и задеплоить смарт-контракты моделирующие процессы покупки и обмена ценными бумагами (возможно выполнение индивидуального варианта).
2. Написать unit-тесты.
3. Продемонстрировать работу через встроенные инструменты Remix IDE.

**Выполнение задания**

Было разработан и написан смарт-контракт на языке Solidity и написаны тесты для него. Код представлен на рисунках 6.1-6.9

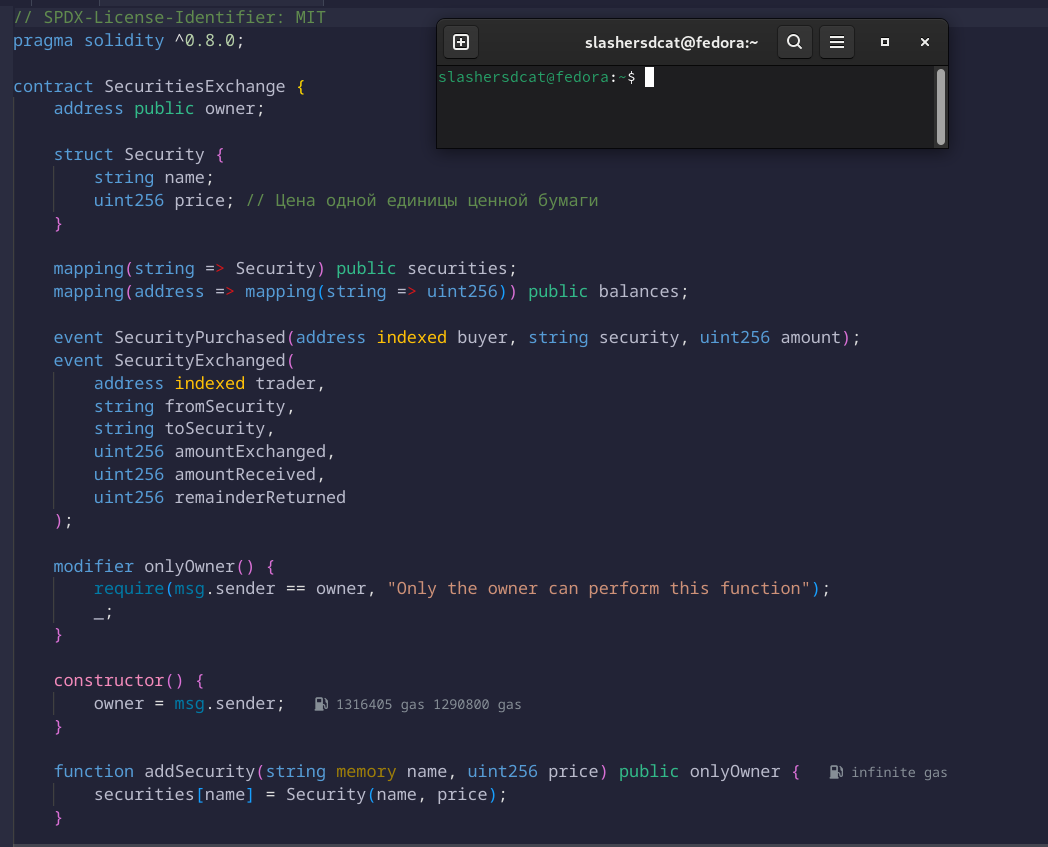


Рисунок 6.1 – Задание 1

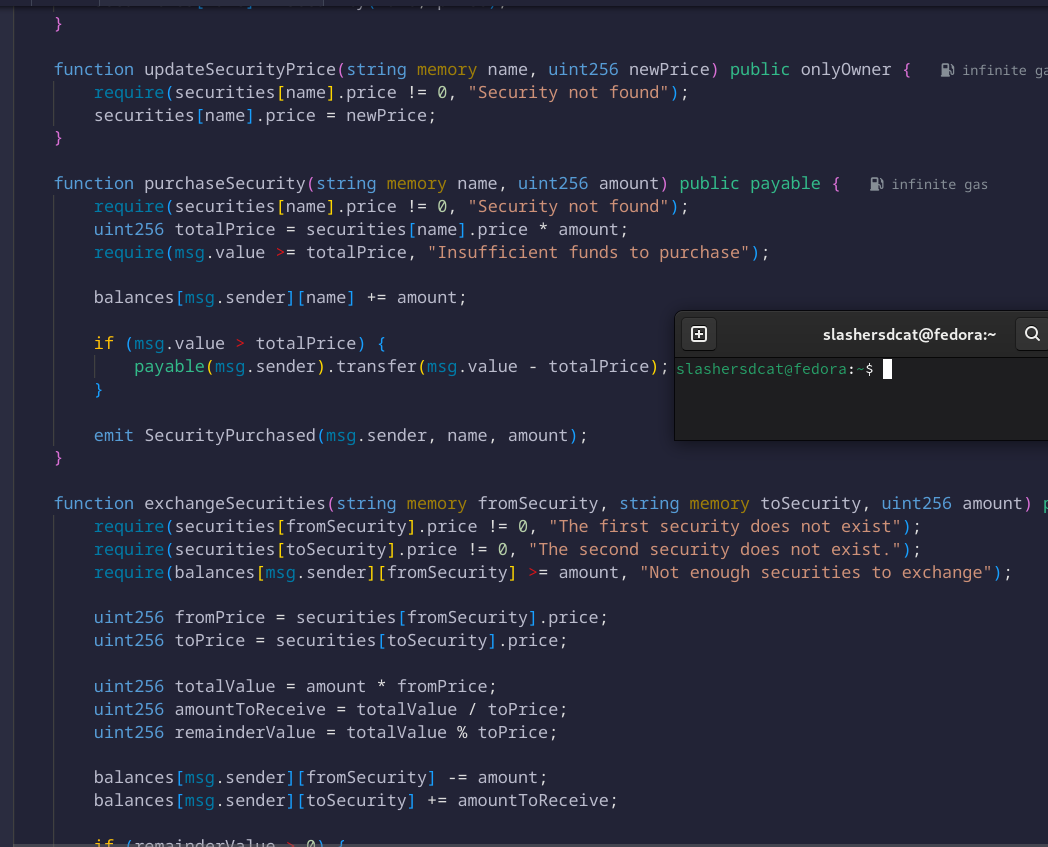


Рисунок 6.2 – Задание 1

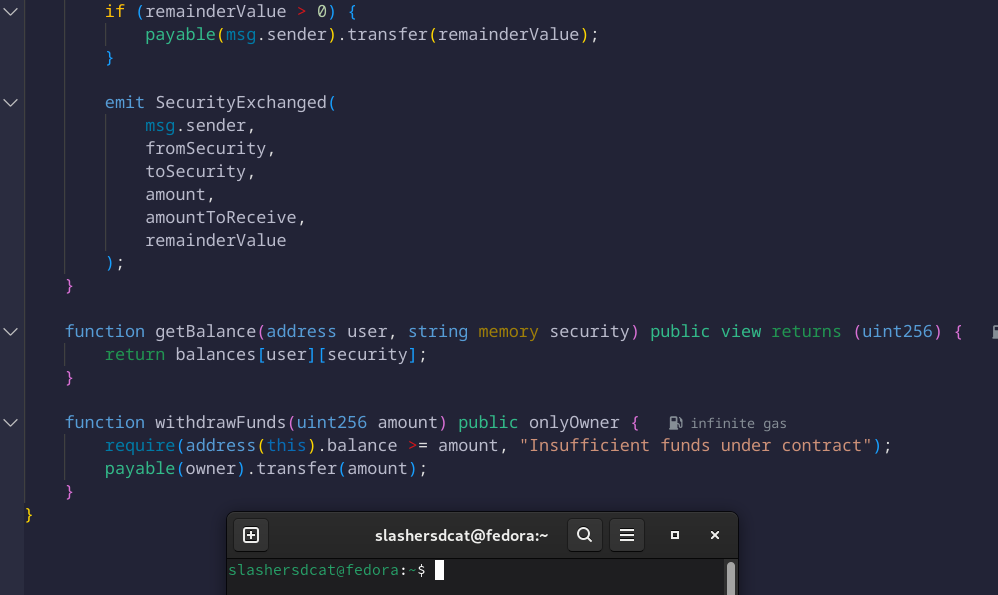


Рисунок 6.3 – Задание 1

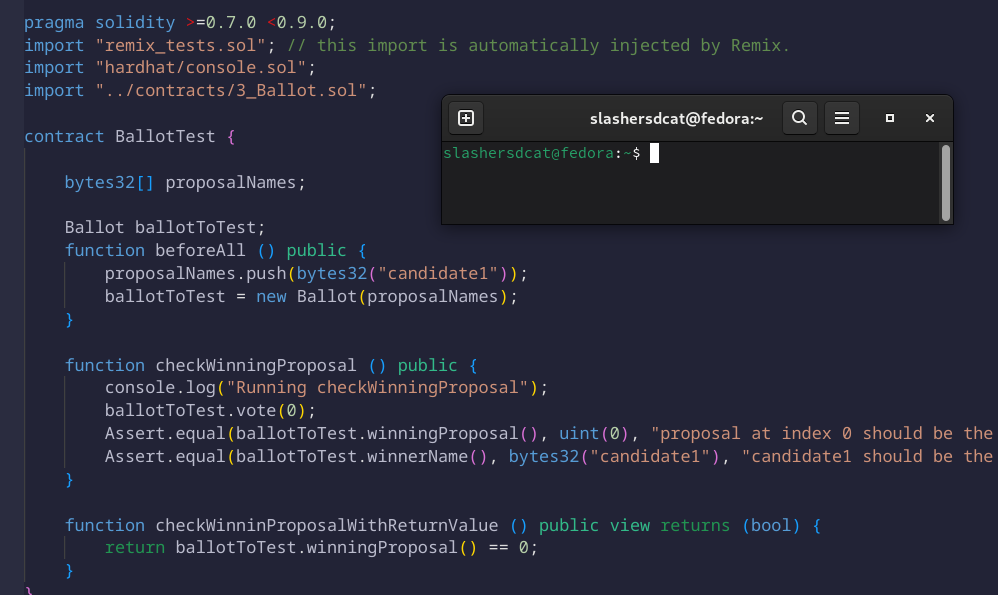


Рисунок 6.4 – Задание 1

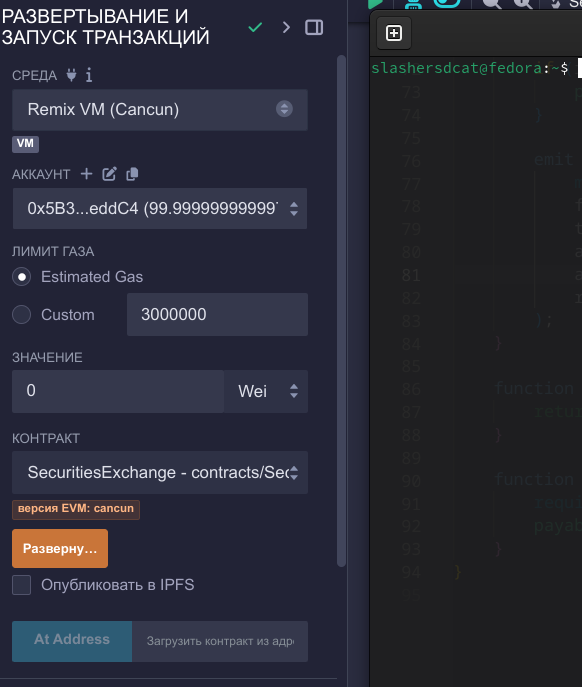


Рисунок 6.5 – Задание 1

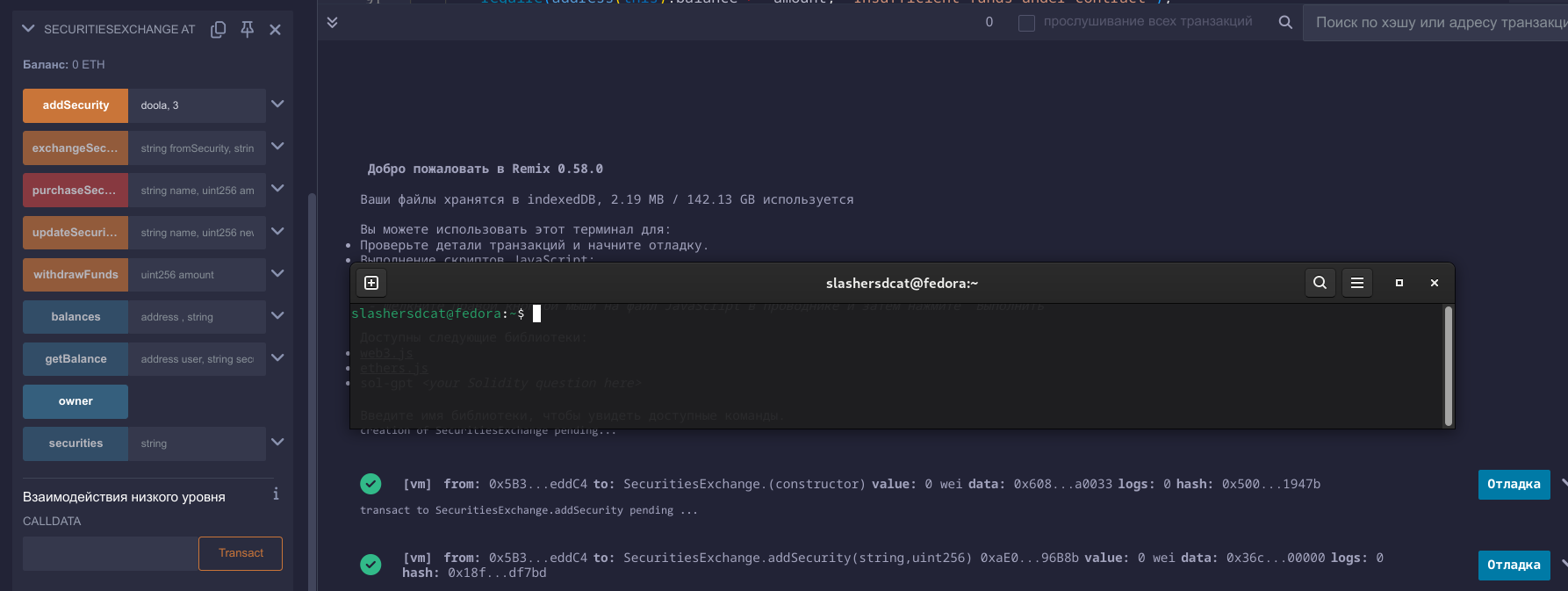


Рисунок 6.6 – Задание 1

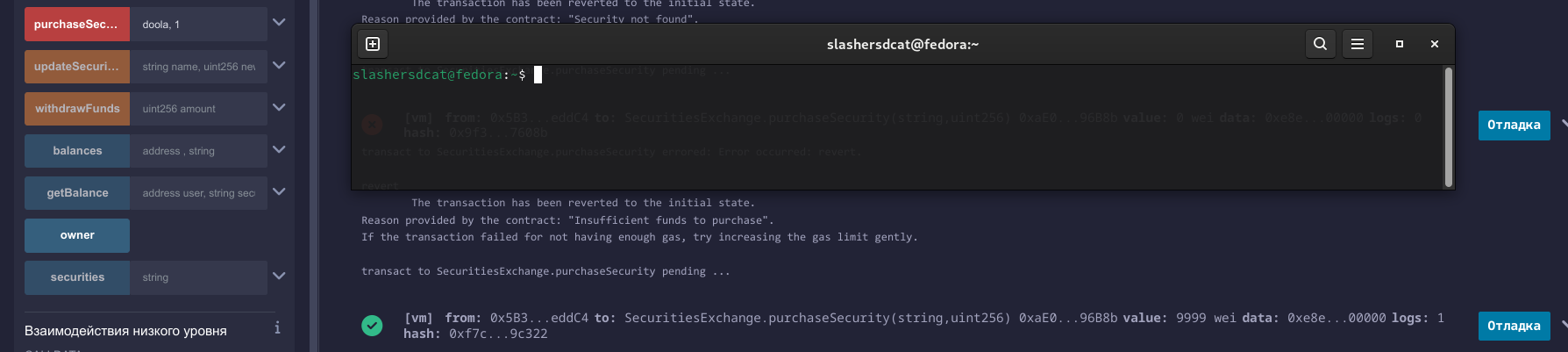


Рисунок 6.7 – Задание 1

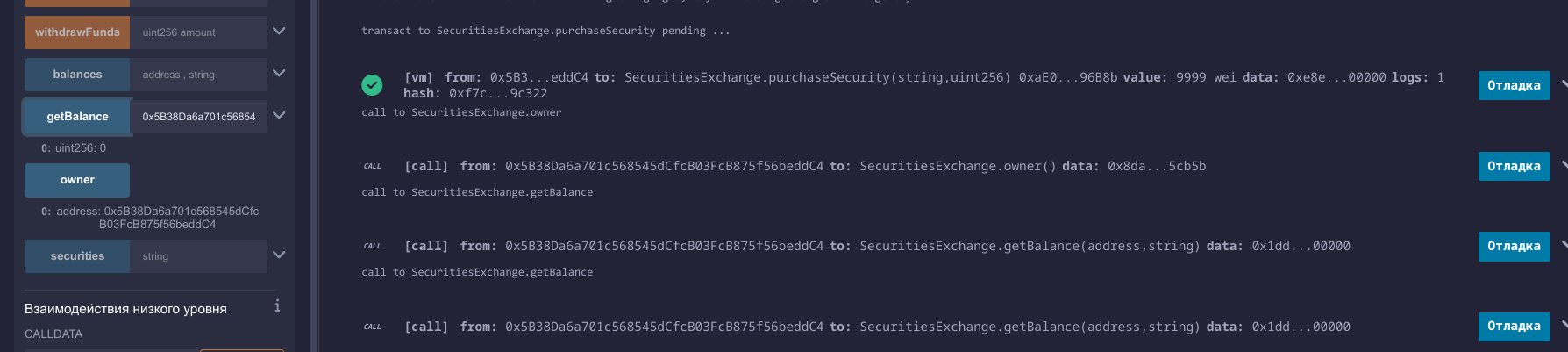


Рисунок 6.8 – Задание 1

# **ПРАКТИЧЕСКАЯ РАБОТА №7**

Данная практическая работа в полном объеме загружена на GitHub и доступна для просмотра по данной ссылке - <https://github.com/SlasherSDCaT/MireaSem7/tree/main/rksp/pr7>

**Задание**

Реализовать сервис, предоставляющий реактивный REST API с использо ванием Spring WebFlux. Предметная область выбирается студентом самостоятельно, но должна быть уникальна в рамках учебной группы.

Требования к сервису:

* Представлено как минимум 5 endpoint’ов;
* Представлены endpoint’в возвращающие как Mono<>, так и Flux<>;
* Продемонстрирована работа с базами данных;
* Продемонстрировано применение операторов преобразования потоков;
* Продемонстрирована работа в формате backpressure;
* Продемонстрирована обработка ошибок в потоках;
* Сервис покрыт unit-тестами.

**Выполнение задания**

Было разработано приложение с помощью фреймворка Spring и протестировано при помощи Insomnia. Код представлен на рисунках 7.1-7.8

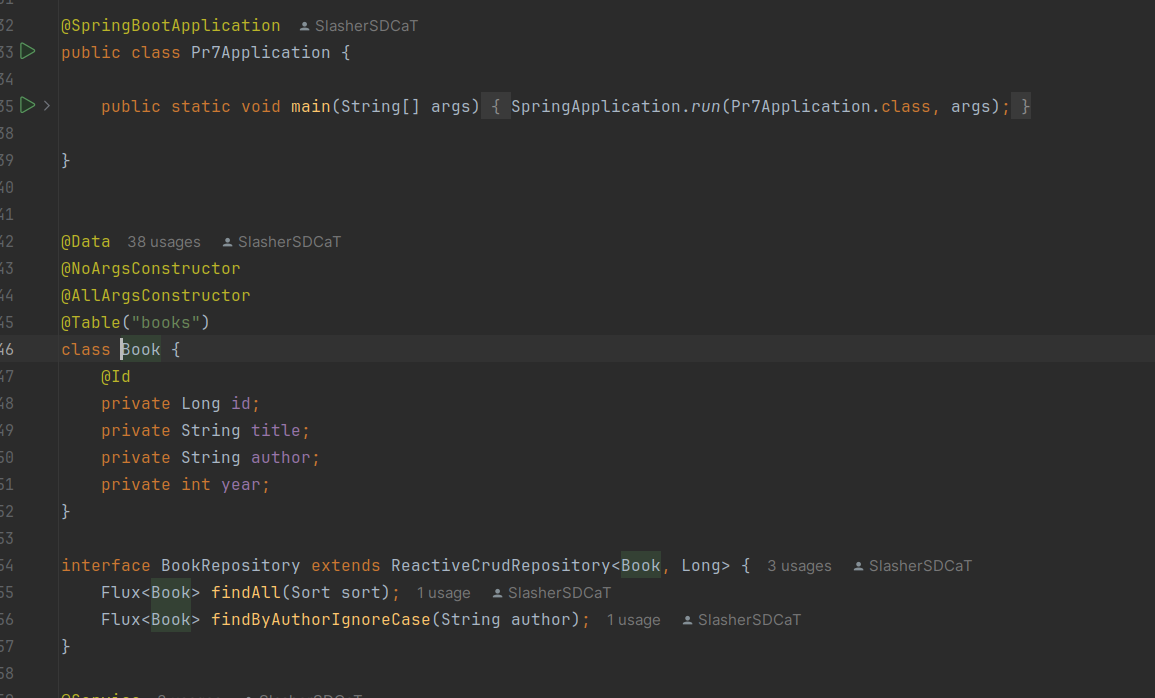


Рисунок 7.1 – Код задания

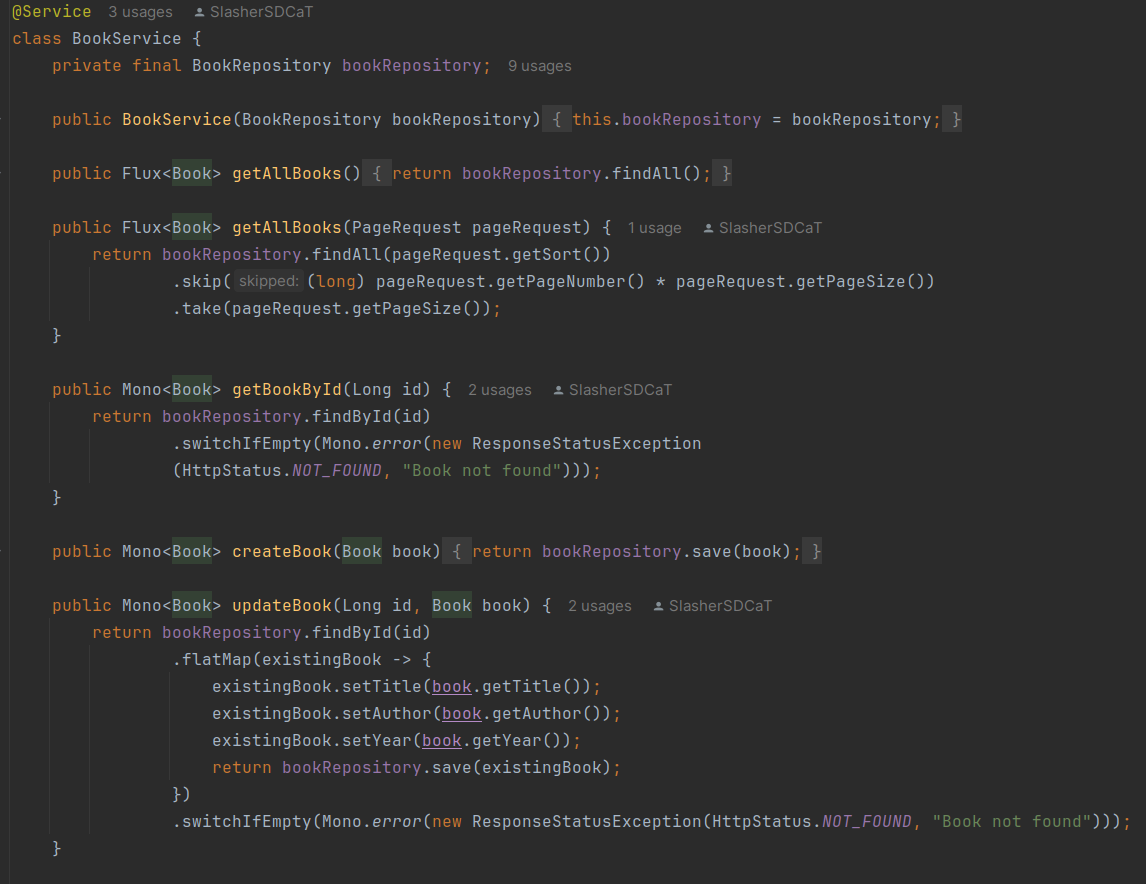


Рисунок 7.2 – Код задания



Рисунок 7.3 – Код задания

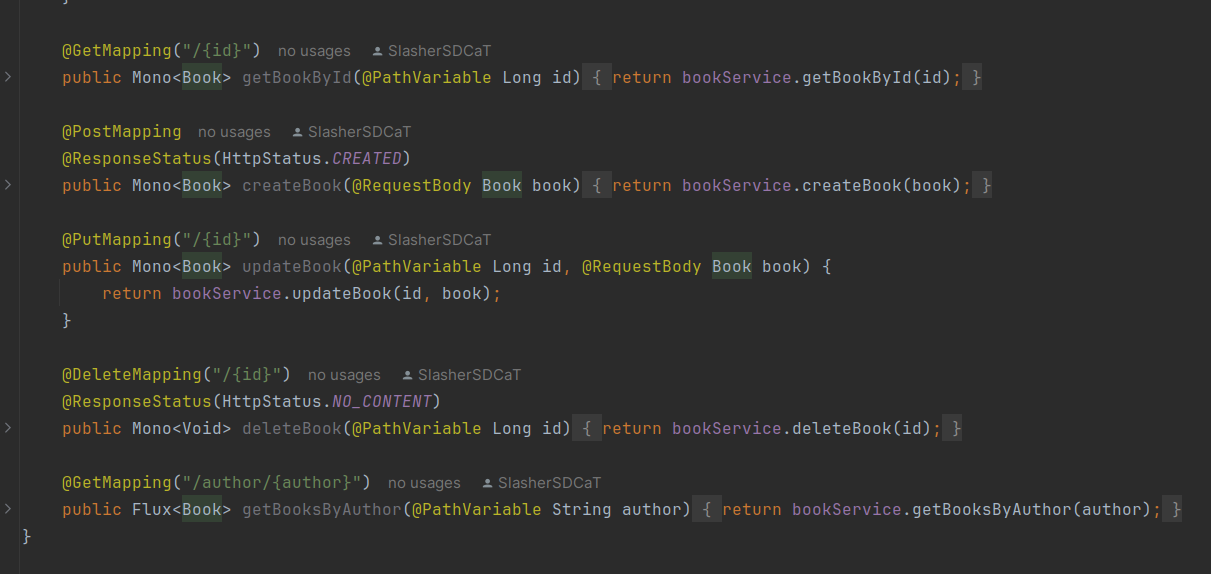


Рисунок 7.4 – Код задания

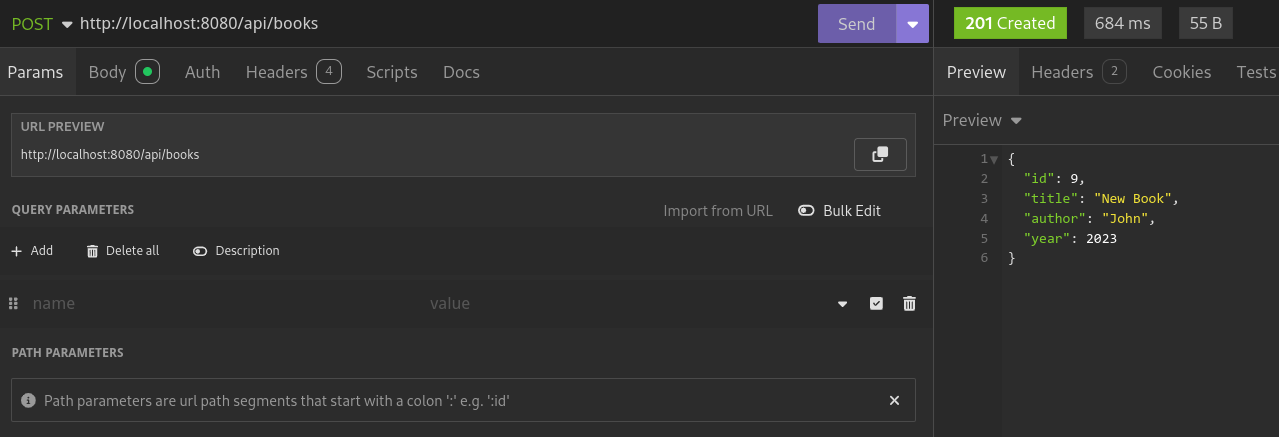


Рисунок 7.5 – Создание книги

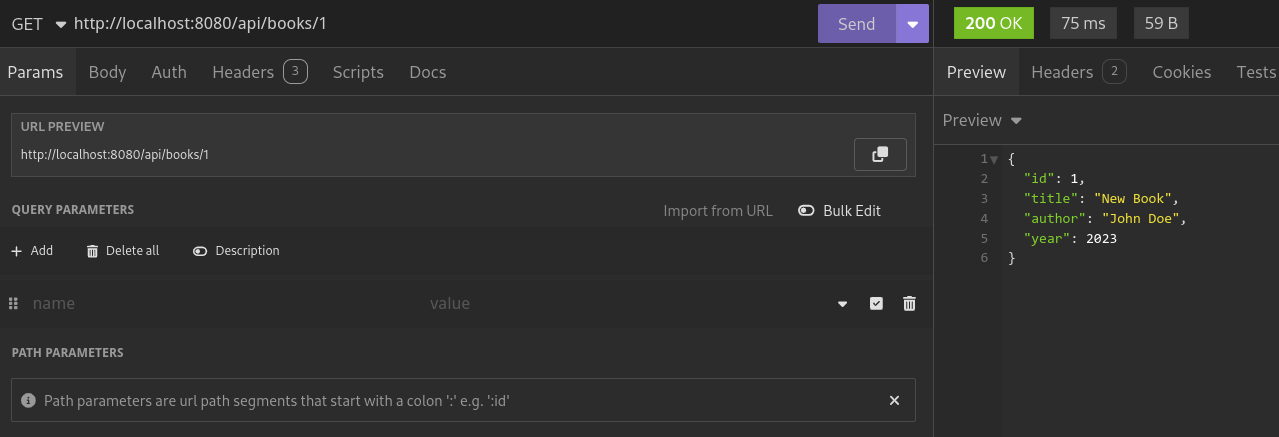


Рисунок 7.6 – Получение по id

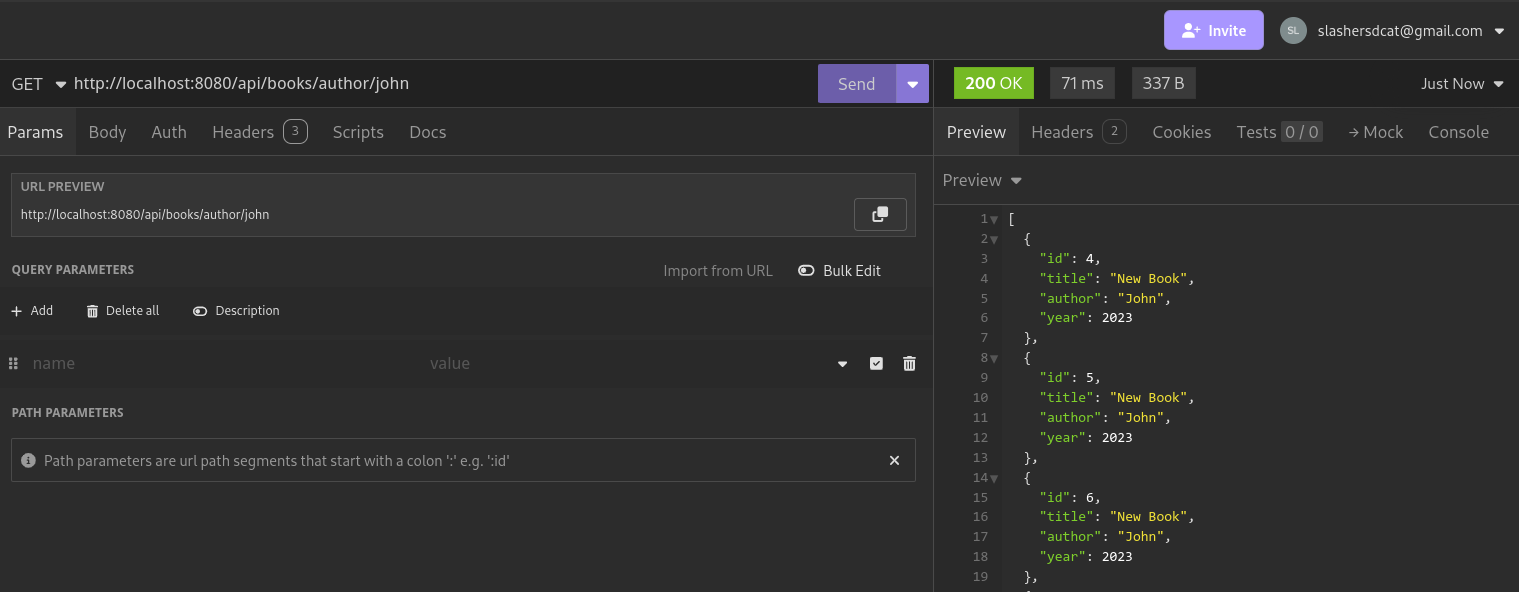


Рисунок 7.7 – Получение по автору

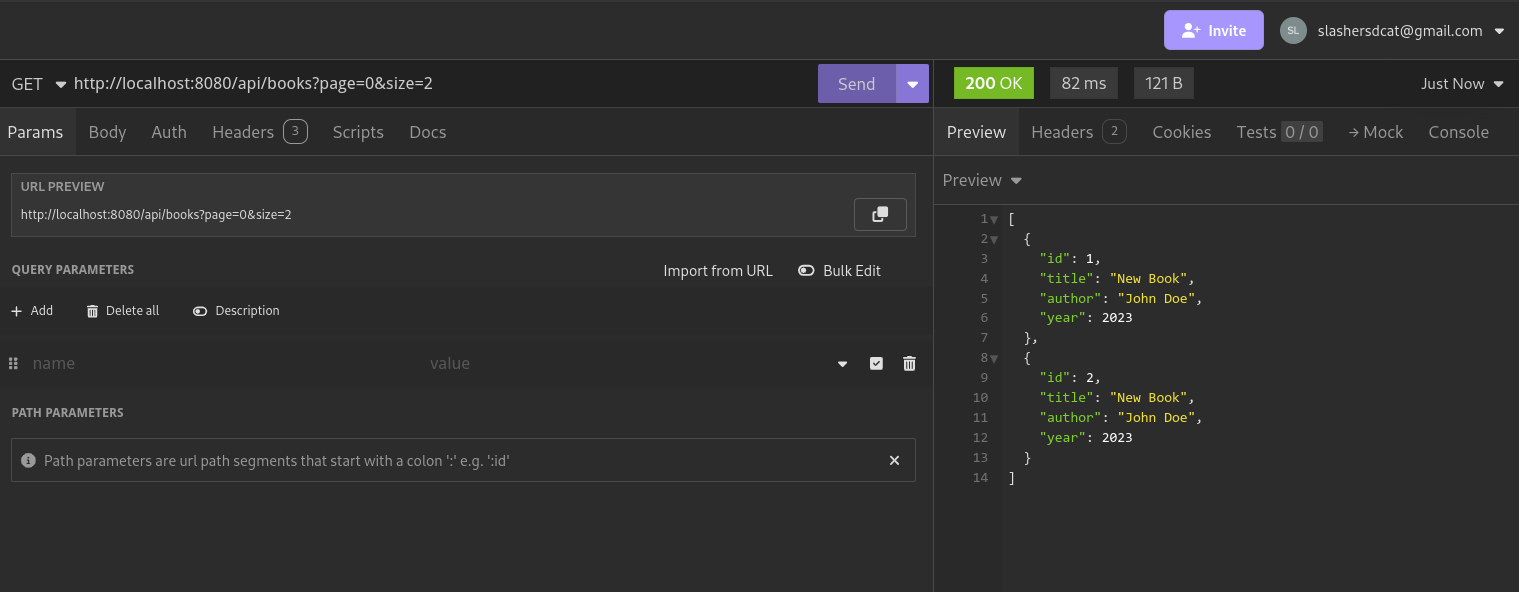


Рисунок 7.8 – Получение с пагинацией

# **ПРАКТИЧЕСКАЯ РАБОТА №8**

Данная практическая работа в полном объеме загружена на GitHub и доступна для просмотра по данной ссылке - <https://github.com/SlasherSDCaT/MireaSem7/tree/main/rksp/prac8>

**Цель работы**

Развить у студентов навыки проектирования и реализации микросервисных систем с использованием инструментов Spring Cloud. Студенты научатся обеспечивать безопасное взаимодействие сервисов, управлять конфигурациями, применять паттерны API Gateway и Service Discovery, балансировать нагрузку, и взаимодействовать с базами данных. Они также освоят развертывание и тестирование микросервисов в облачных средах или локально.

**Задание на практическую работу**

Разработать микросервисную систему с использованием инструментов Spring Cloud (возможно использование аналогов). Система должна удовлетворять следующим требованиям:

* реализована авторизация при помощи протокола OAuth2;
* реализован паттерн API Gateway;
* реализован паттерн Service Discovery;
* реализован балансировщик нагрузки;
* реализовано как минимум 3 микросервиса, связанных с предметной областью;
* продемонстрировано взаимодействие между микросервисами;
* микросервисы взаимодействуют с базой данных;
* для каждого микросервиса используется отдельный экземпляр базы данных;

**Выполнение задания**

Было написано 4 микросервиса – gateway, сервис уведомлений, сервис задач, сервис авторизации. Код сервисов предоставлен на рисунке 8.1-8.9

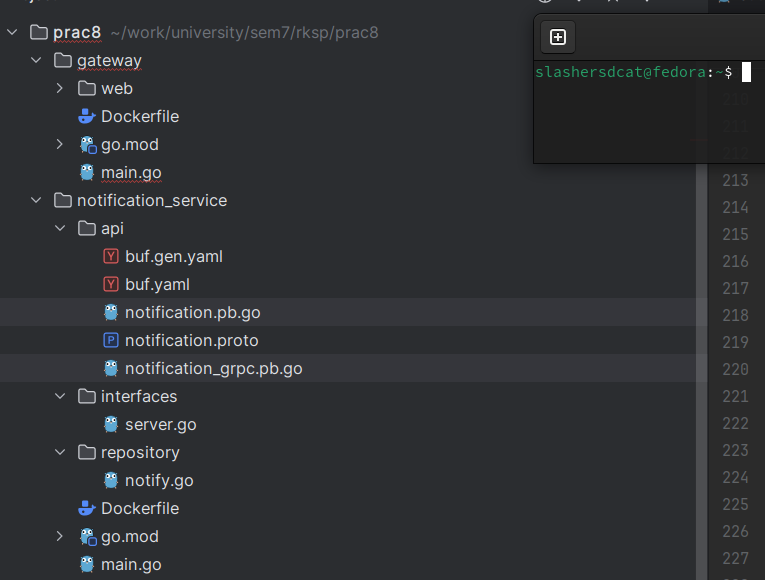


Рисунок 8.1 – архитектура сервисов gateway и уведомлений

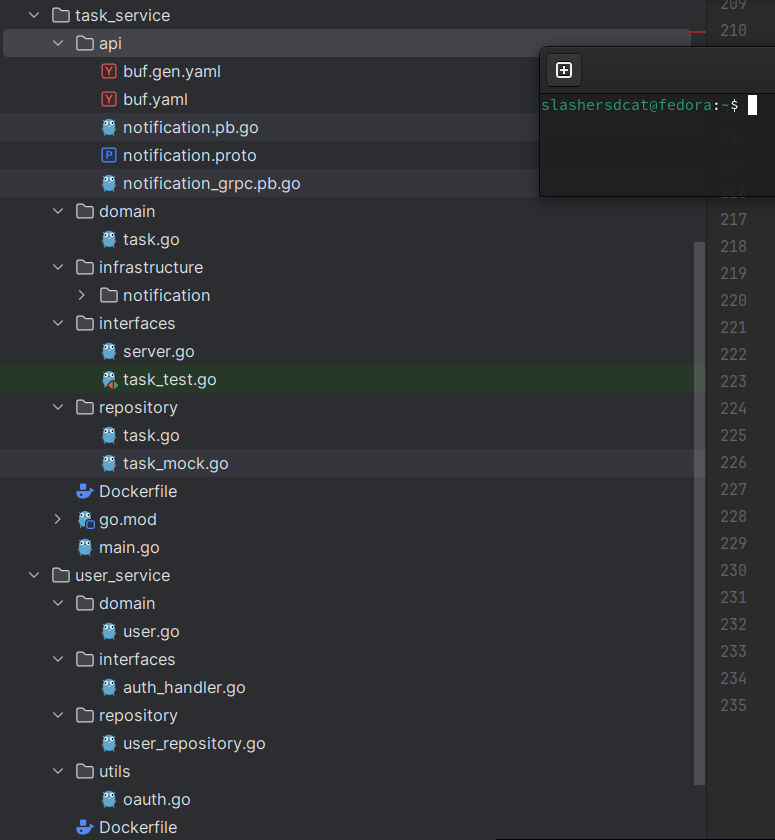


Рисунок 8.2 – архитектура сервисов задач и авторизации

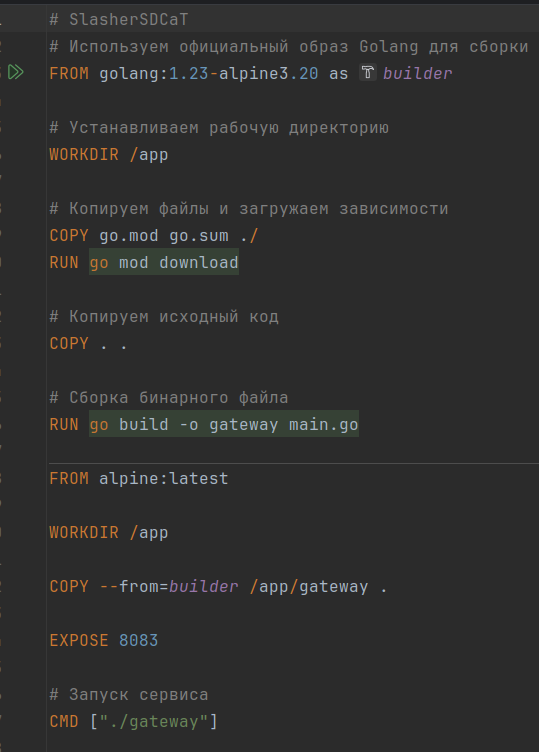


Рисунок 8.3 – Dockerfile gateway



Рисунок 8.4 – dockerfile сервиса уведомлений

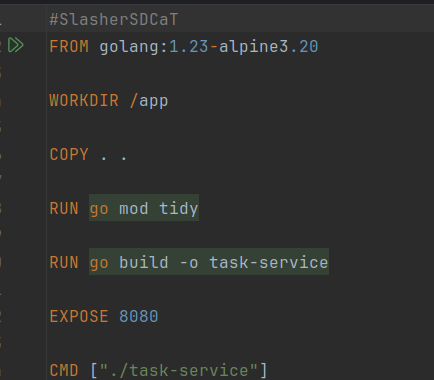


Рисунок 8.5 – dockerfile сервиса заданий

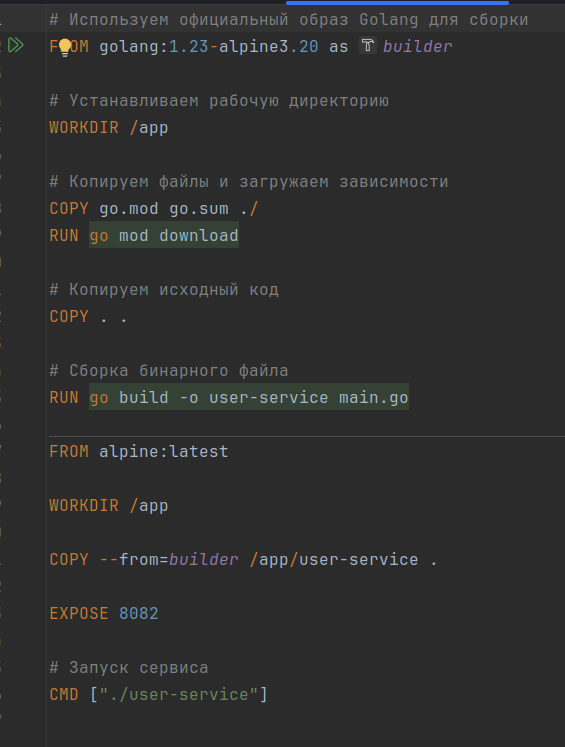


Рисунок 8.6 – Dockerfile сервиса авторизации



Рисунок 8.7 – docker-compose поднятие сервиса gateway и сервиса заданий с базой данных для данного сервиса



Рисунок 8.8 – docker-compose поднятие сервиса уведомлений и базы данных для данного сервиса



Рисунок 8.9 – docker-compose для сервиса авторизации и бд для данного сервиса

Далее приложение было развернуто и протестирован его функционал. Процесс отображен на рисунках 8.10-8.12

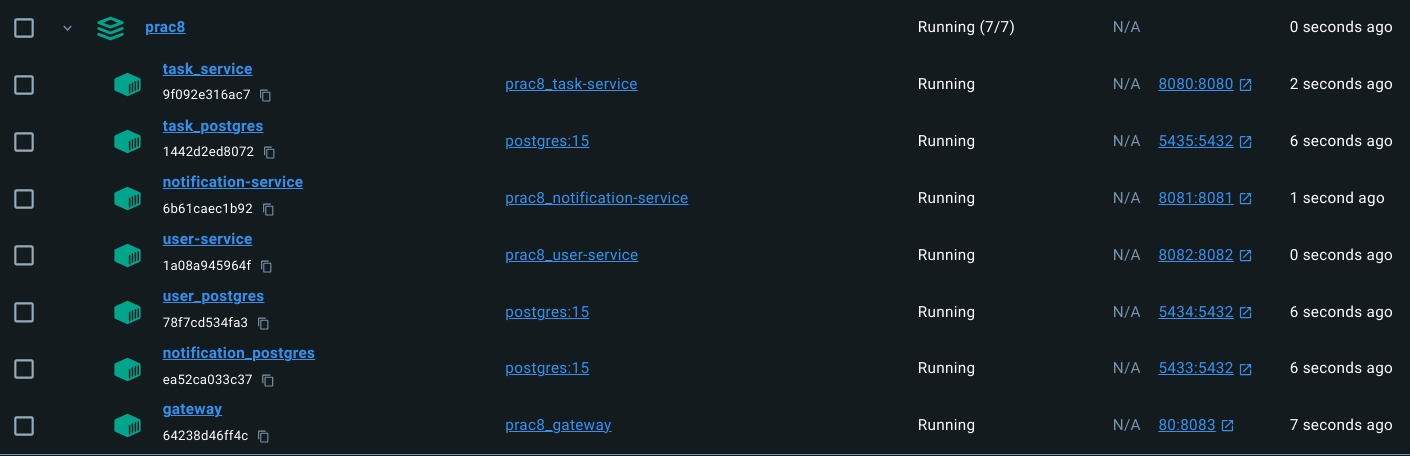


Рисунок 8.10 – поднятые микросервисы

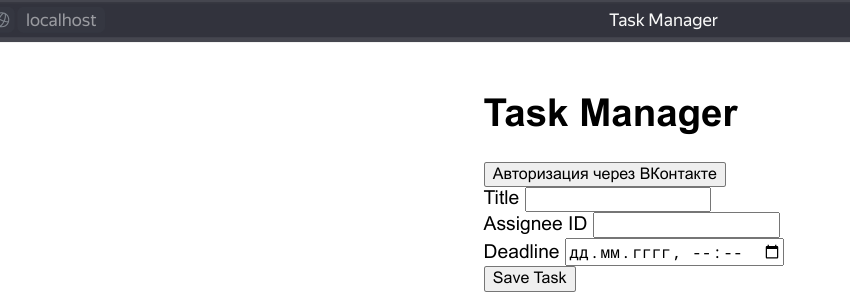


Рисунок 8.11 – клиентская часть

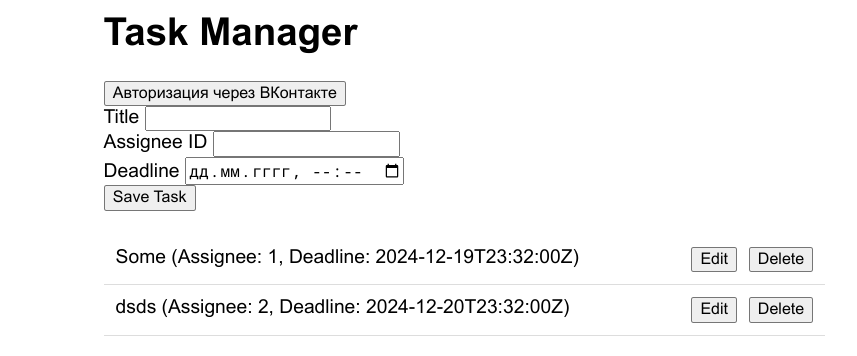


Рисунок 8.12 – авторизация через VKid и добавление задач

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. K8S для начинающих. Первая часть — Текст: электронный [сайт]. — URL: <https://habr.com/ru/post/589415/>
2. KrakenD Designer | KrakenD — Текст: электронный [сайт]. — URL: <https://designer.krakend.io/>
3. Operator for Kubernetes | Jaeger — Текст: электронный [сайт]. — URL: <https://www.jaegertracing.io/docs/1.24/operator/>
4. Kafka 3.3 Documentation | Kafka — Текст: электронный [сайт]. — URL: <https://kafka.apache.org/documentation/>
5. Helm Docs | Helm — Текст: электронный [сайт]. — URL: <https://helm.sh/docs/>
6. KrakenD Documentation | KrakenD — Текст: электронный [сайт]. — URL: <https://www.krakend.io/docs/overview/>
7. 50 вопросов по Docker, которые задают на собеседованиях, и ответы на них | Хабр. — Текст: электронный [сайт]. — URL: https://habr.com/ru/company/southbridge/blog/528206/
8. Docker Documentation | Docker Documentation — Текст: электронный [сайт]. — URL: — Текст: электронный [сайт]. — URL: <https://docs.docker.com/>
9. Zabbix Documentation — Текст: электронный [сайт]. — URL: <https://www.zabbix.com/manuals>
10. Prometheus Documentation — Текст: электронный [сайт]. — URL: <https://prometheus.io/docs/introduction/overview/>
11. Grafana Documentation — Текст: электронный [сайт]. — URL: <https://grafana.com/docs/>
12. GrayLog Documentation — Текст: электронный [сайт]. — URL: https://docs.graylog.org/