**Разработка Клиент Серверных приложений**

**Сидоров С.Д.**

**ИКБО-20-21**

# Практическая работа 1

**Задание 1**

Был создан массив из 10000 случайных чисел.

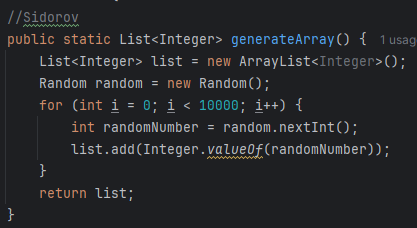


Рисунок 1 – Функция создания списка из 10000 элементов

В созданном массиве был начат поиск наименьшего элемента последовательно, многопоточно и с использованием Fork Join.

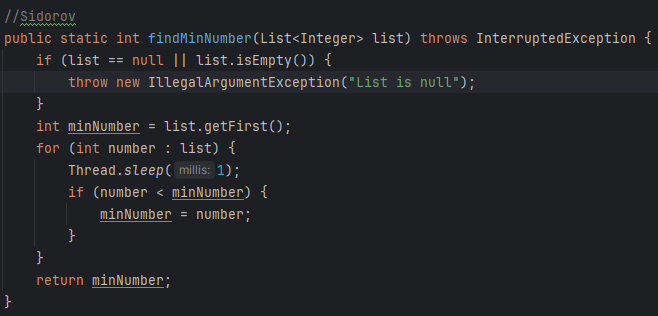


Рисунок 2 – Последовательный способ

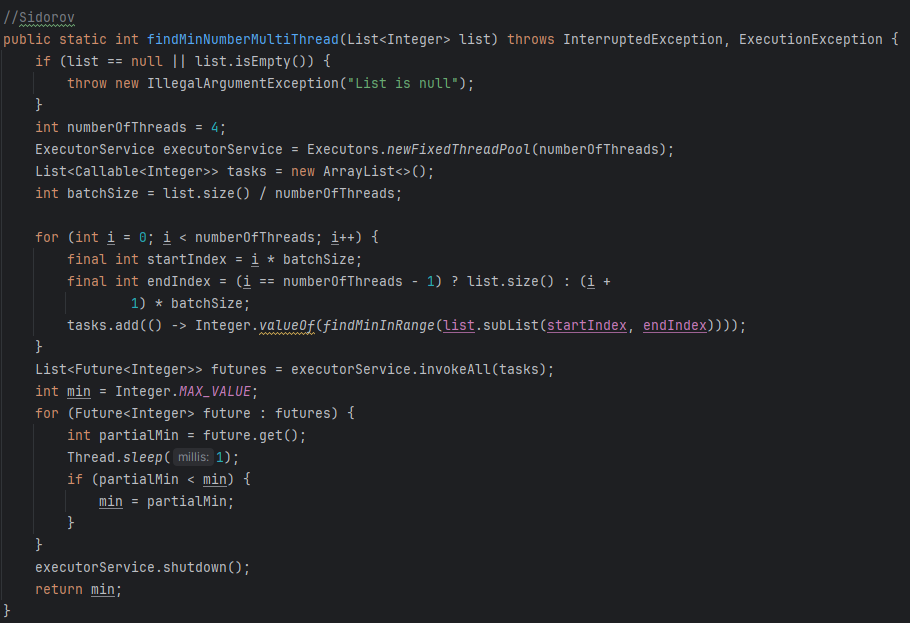


Рисунок 3 – Многопоточный способ

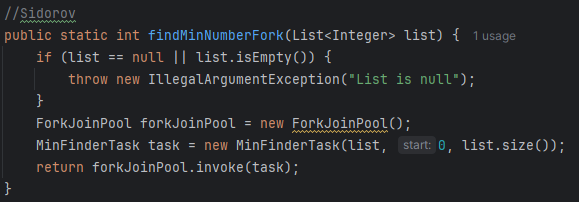


Рисунок 4 – Fork

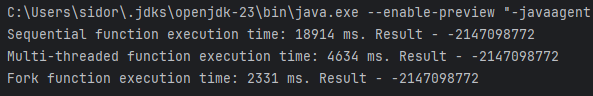


Рисунок 5 – Результат

Последовательный поиск наименьшего элемента оказался самым медленным.

**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.

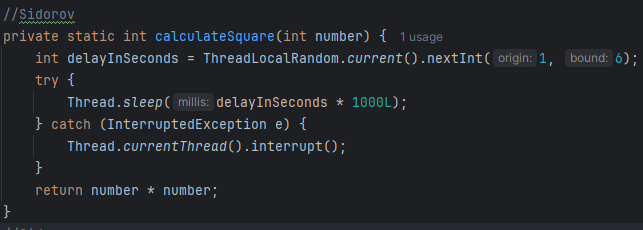


Рисунок 6 – Функция, высчитывающая квадрат

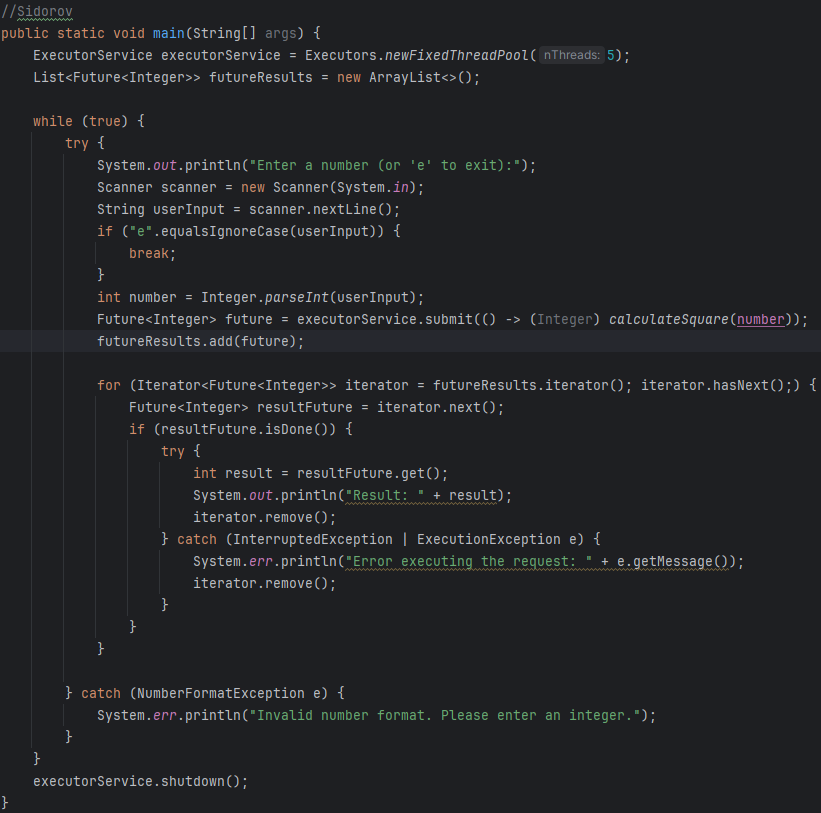


Рисунок 7 – Функция, обрабатывающая значения пользователя

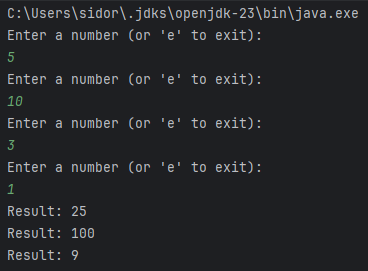


Рисунок 8 – Результат

В момент выполнения запроса пользователь имеет возможность отправить новый запрос.

**Задание 3**

Класс 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».

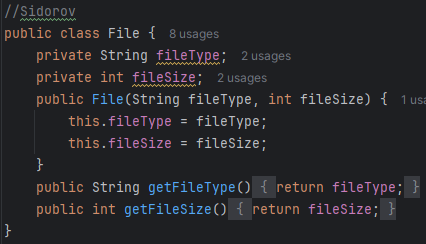


Рисунок 9 – Класс File

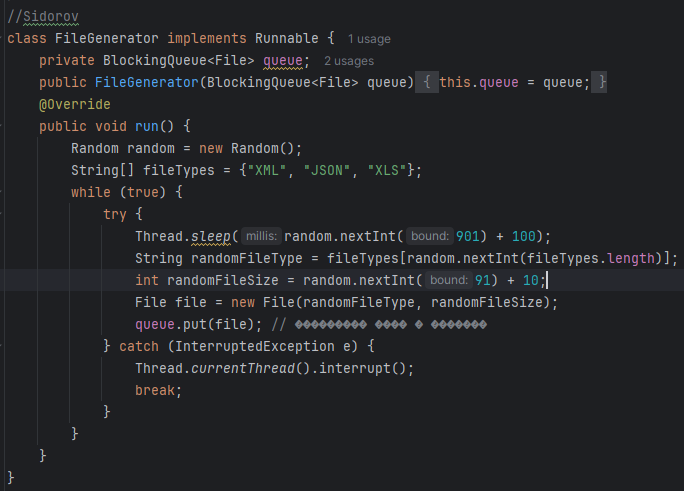


Рисунок 10 – Класс FileGenerator

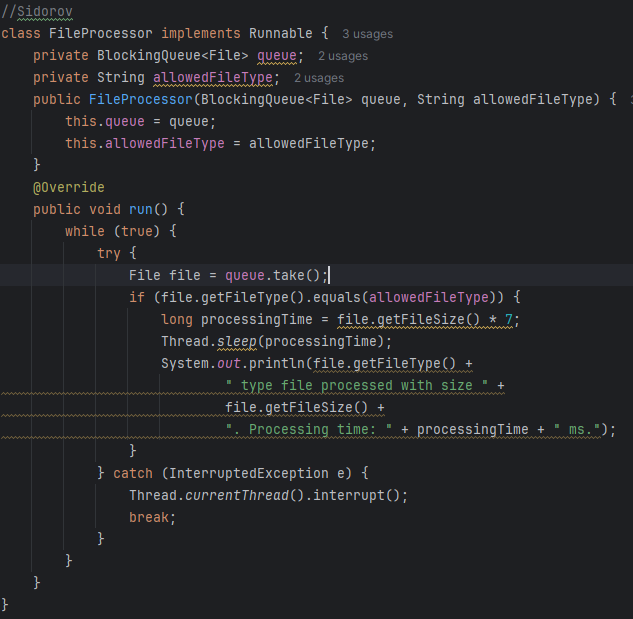


Рисунок 11 – Класс FileProcessor

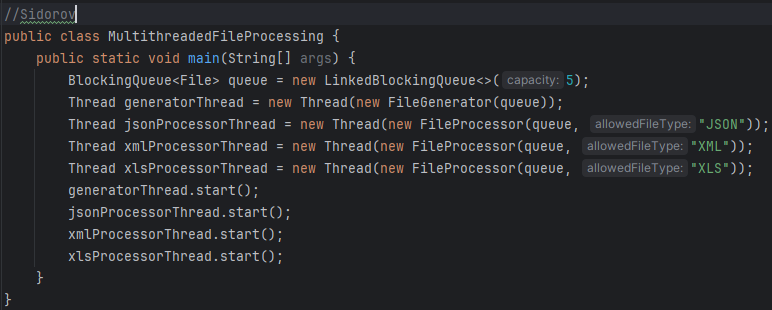


Рисунок 12 – Запуск потоков

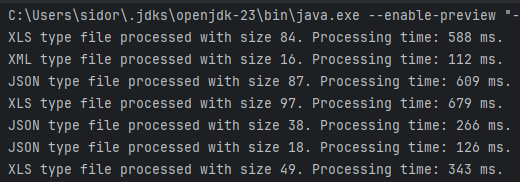


Рисунок 13 – Результат

В результате, обеспечена потокобезопасность, работа генератора не зависит от работы обработчиков, потоки не блокируют другие потоки при отсутствии задач

# Практическая работа №2

**Задание 1.**

Для выполнения задания были созданы функции отвечающие за создание и чтение файла. Код представлен на рисунке 14.

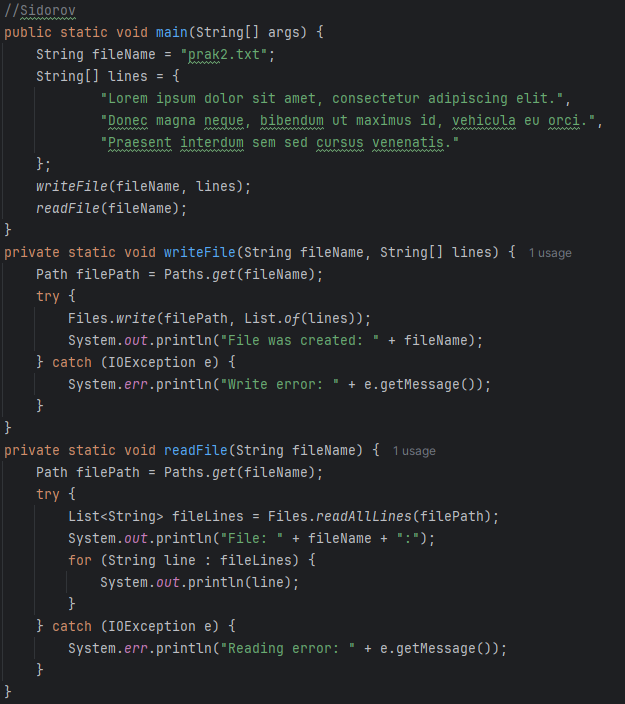


Рисунок 14 – Функции осуществляющие операции с файлом

Результат выполнения представлен на рисунке 15.

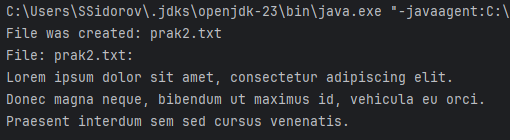


Рисунок 15 – результат вывода содержимого файла

**Задание 2.**

Для выполнения необходимых замеров и самих операций был написан метод main, представленный на рисунке 16.

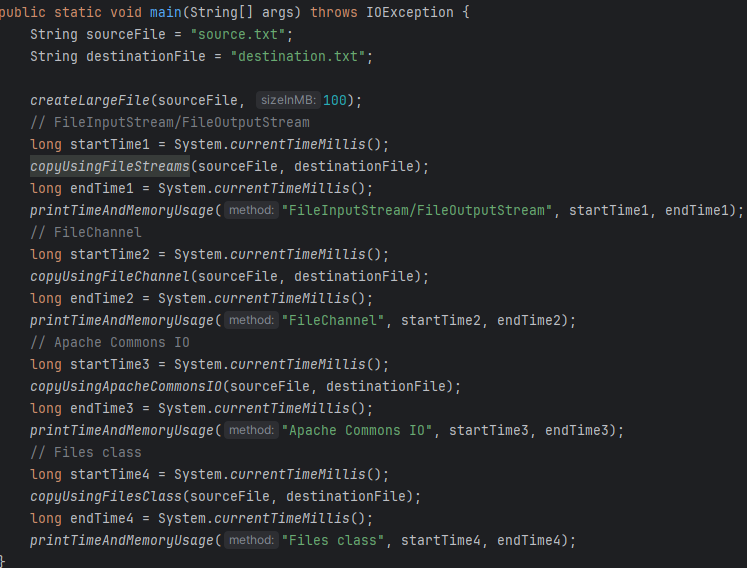


Рисунок 16 – Основной метод программы

В данном методе вызываются методы, отвечающие за выполнение копирования. Код методов представлен на рисунке 17.

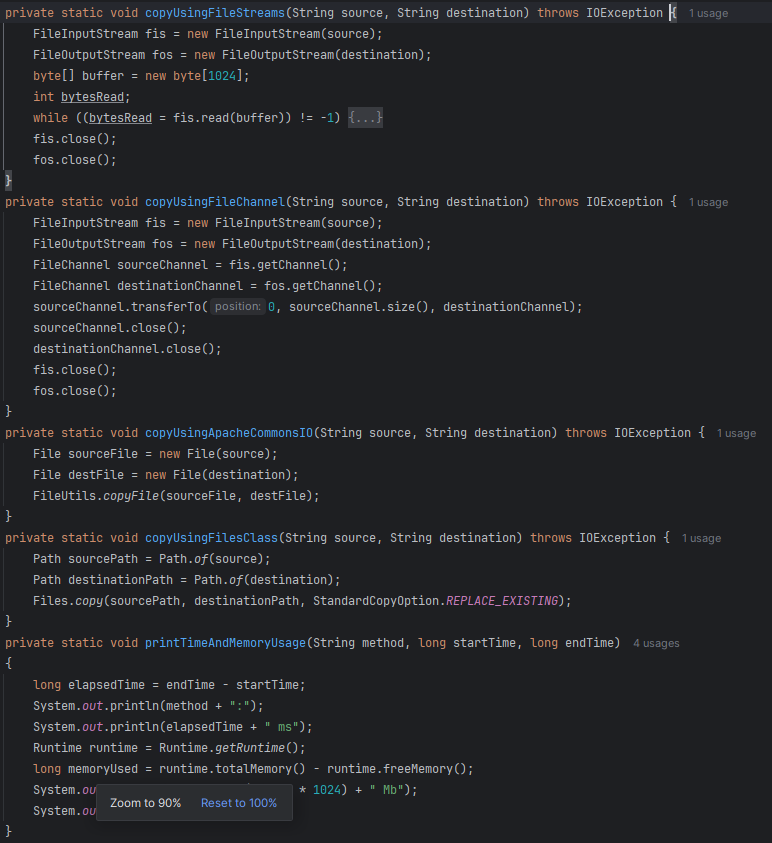


Рисунок 17 – Методы, отвечающие за копирование файла

Результат выполнения представлен на рисунке 18.

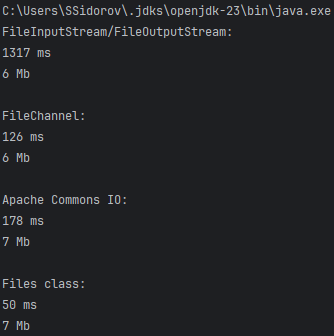


Рисунок 18 – Результат выполнения

**Задание 3.**

Для выполнения данного задания был написан метод calculateChecksum, вычисляющий 16-битную контрольную сумму файла, код которого представлен на рисунке 19.

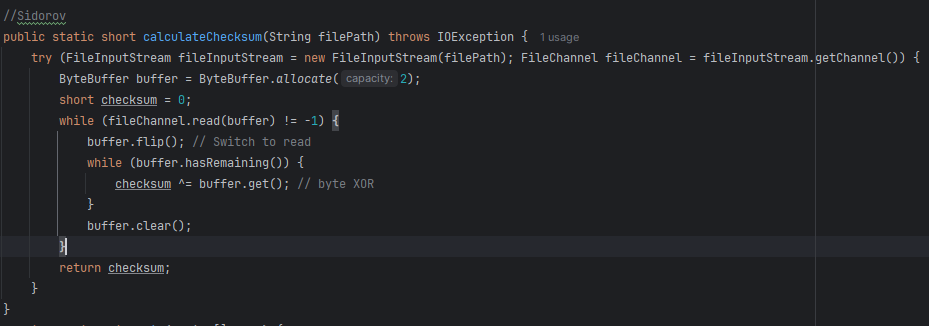


Рисунок 19 – Метод calculateChecksum

Результат вычисления для тестового текстового файла представлен на рисунке 20.

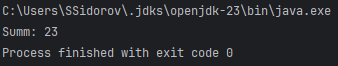


Рисунок 20 – Результат работы метода

**Задание 4**

Для четвертого задания были созданы функции для отслеживания изменений и чтения файла. Данные функции представлены на рисунке 22.

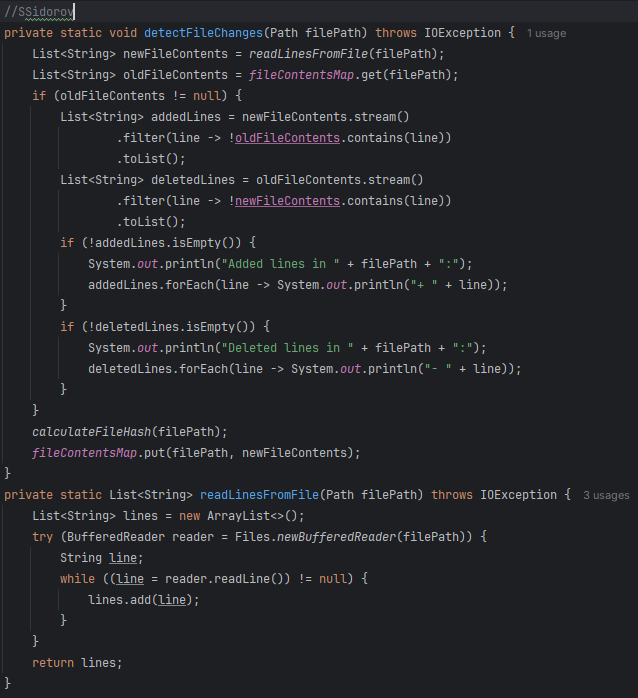


Рисунок 22 – Функции отслеживания изменений и чтения файла

Также была создана функция для подсчёта хэша, представленная на рисунке 23.

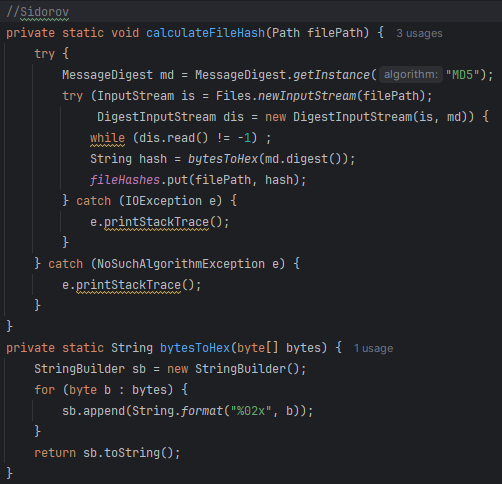


Рисунок 23 – Функция для подсчёта хэша

В функции main() происходит отслеживание изменений указанной директории, а также вызов соответствующих методов. Функция представлена на рисунке 24.

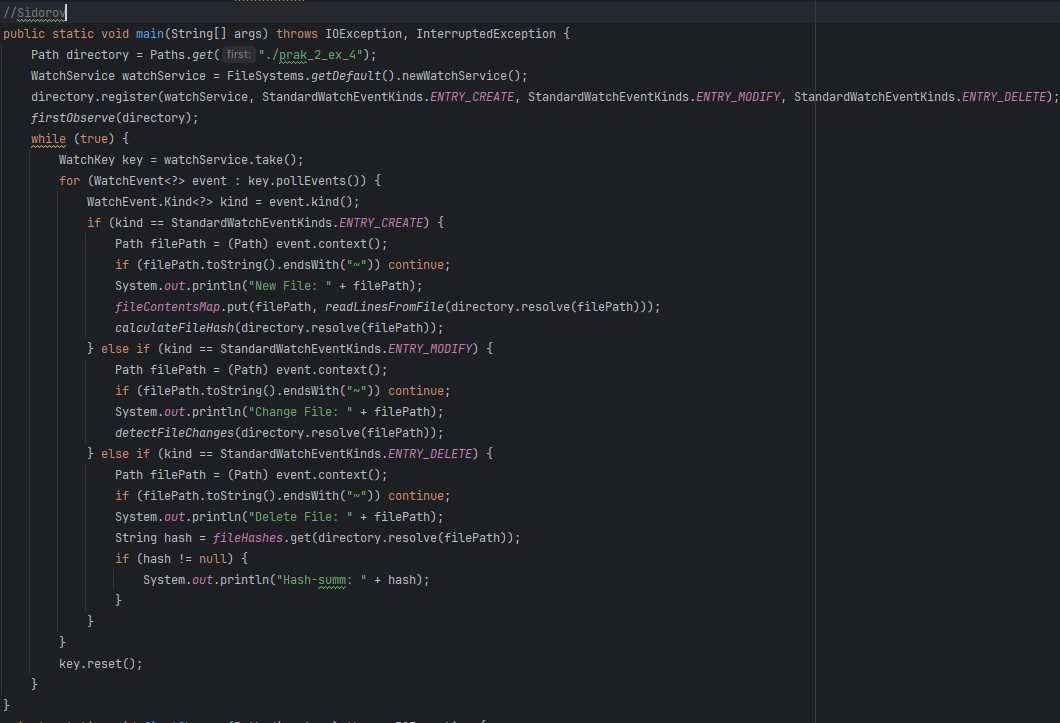


Рисунок 24 – Функция main

Результат работы программы представлен на рисунке 25.

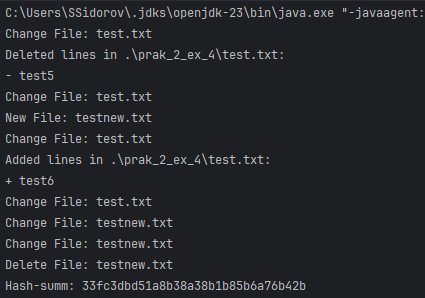


Рисунок 25 – Результат работы

# Практическая работа №3

**Задание 1.**

Для выполнения данного задания был сформирован основной код, отвечающий за создание и организацию работу датчиков и сигнализаций. Код класса представлен на рисунке 26.

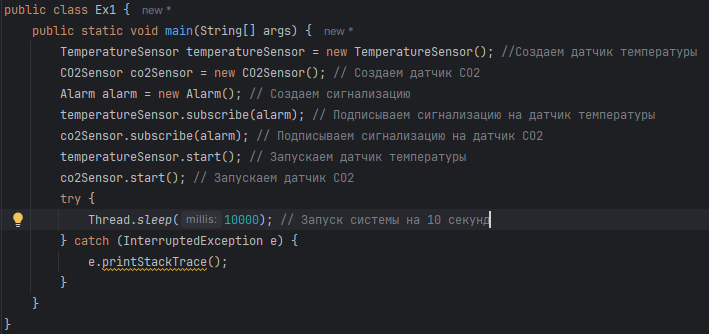


Рисунок 26 – Код основного класса для задания 1

Данный код использует вспомогательные классы, представленные на рисунках 27 – 29.

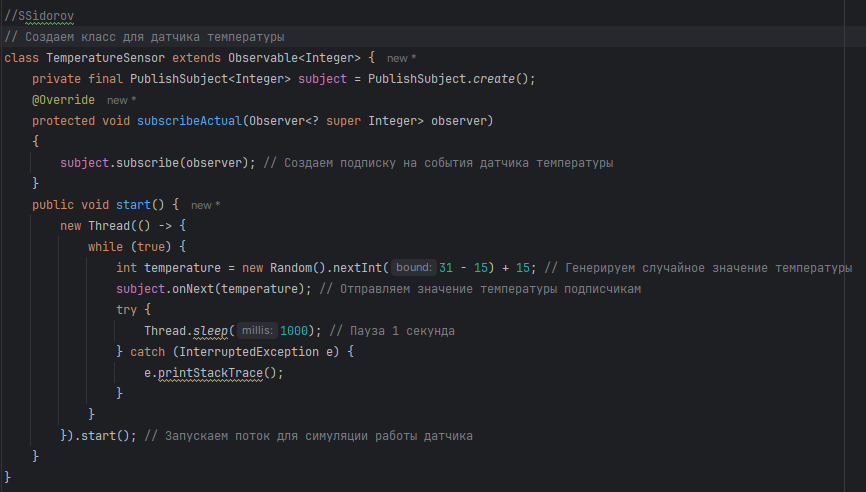


Рисунок 27 – Код для датчика температуры

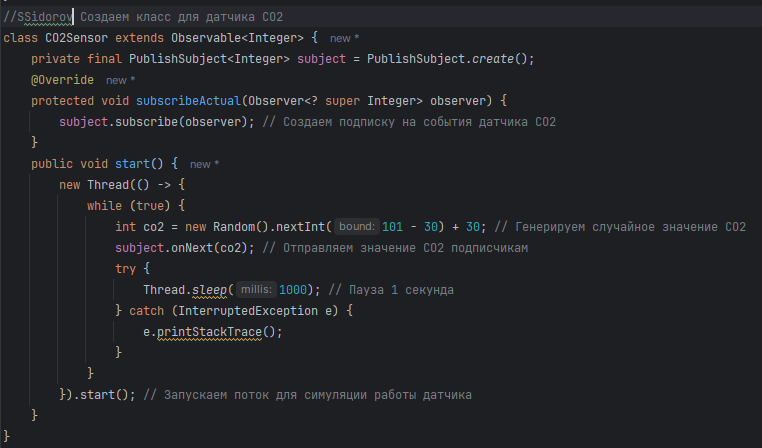


Рисунок 28 – Код для датчика CO2



Рисунок 29 – Код для сигнализации

Результат выполнения представлен на рисунке 30.

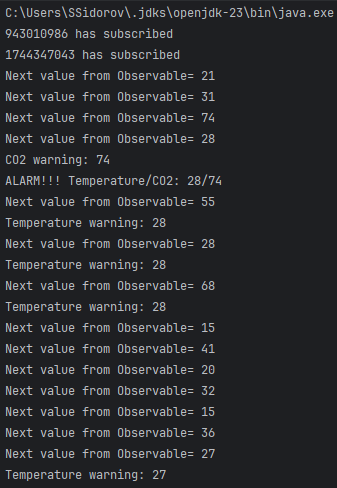


Рисунок 30 – Результат выполнения программы

**Задание 2.**

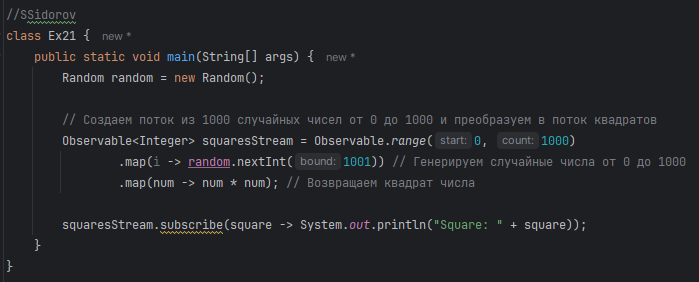
Код для выполнения задания 2.1 представлен на рисунке 31.   


Рисунок 31 – Работа с потоками случайных чисел

Результат выполнения представлен на рисунке 32.

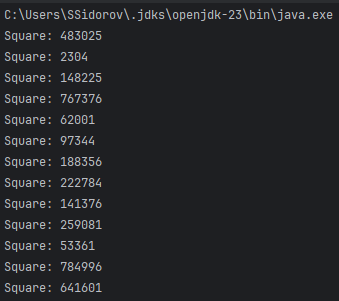


Рисунок 32 – Результат выполнения задания 2.1

Код для выполнения задания 2.2 представлен на рисунке 33.

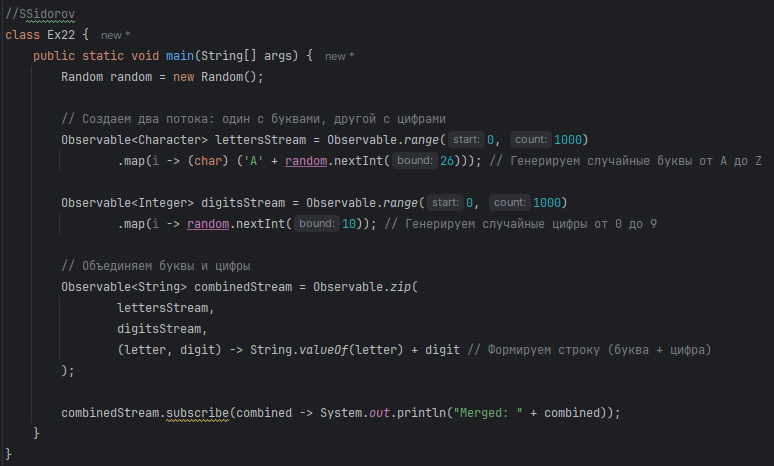


Рисунок 33 – Параллельная обработка потоков

Результат работы представлен на рисунке 34.

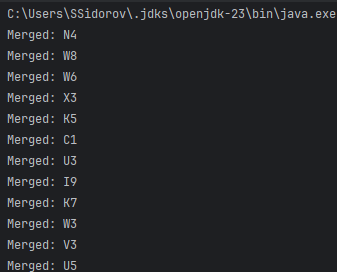


Рисунок 34 – Результат параллельной обработки потоков

Код для выполнения задания 2.3 представлен на рисунке 35.

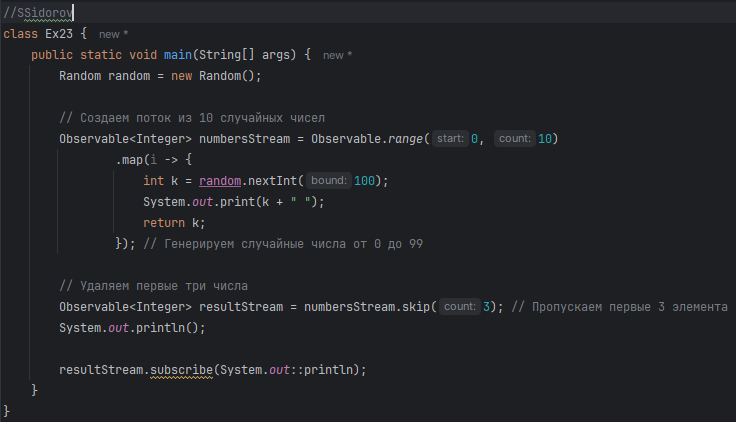


Рисунок 35 – Случайный поток в случайное число

Результат выполнения содержится на рисунке 36.

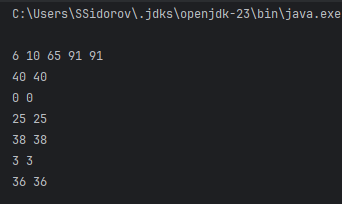


Рисунок 36 – Результат выполнения задания 2.3

**Задание 3.**

Для выполнения 3го задания был создан класс UserFriend, его код представлен на рисунке 37.



Рисунок 37 – Класс UserFriend

Результат выполнения представлен на рисунке 38.

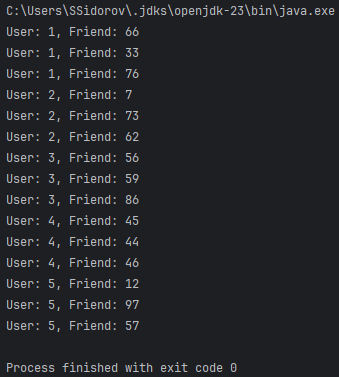


Рисунок 38 – Результат выполнения задания 3

**Задание 4.**

Класс File представлен на рисунке 39.

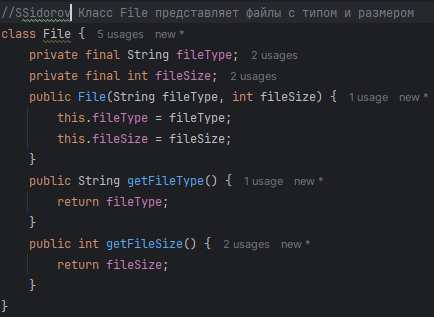


Рисунок 39 – Код класса File

Код класса FileGenerator представлен на рисунке 40.

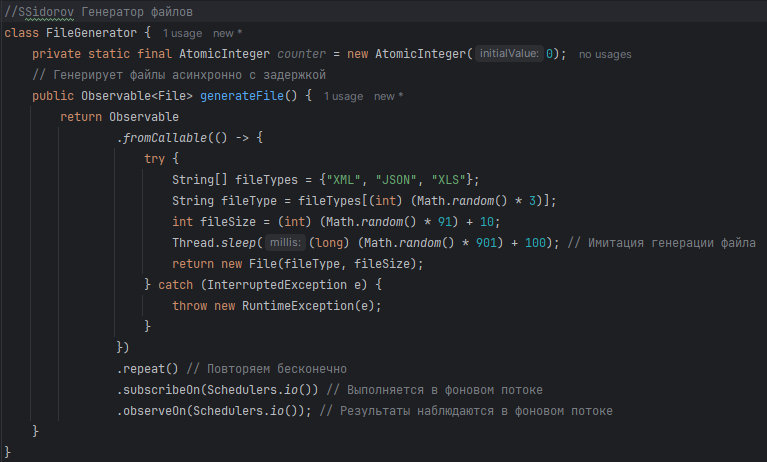


Рисунок 41 – Код FileGenerator

Код класса FileQueue представлен на рисунке 42.

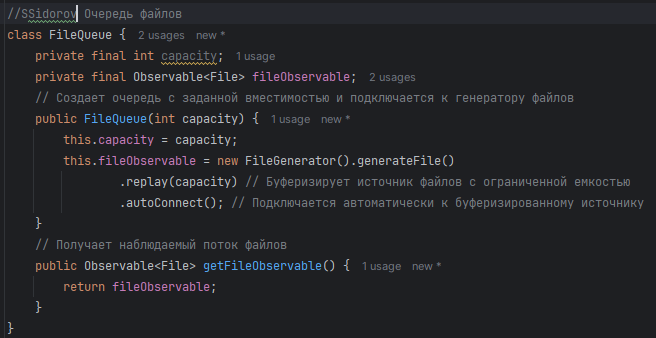


Рисунок 42 – Код FileQueue

Код класса FileProcessor представлен на рисунке 43.

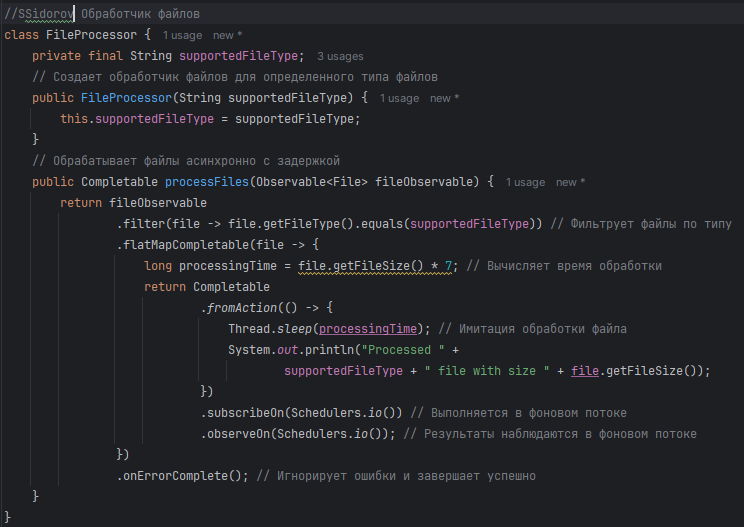


Рисунок 43 – Код FileProcessor

Код основного класса системы представлен на рисунке 44.

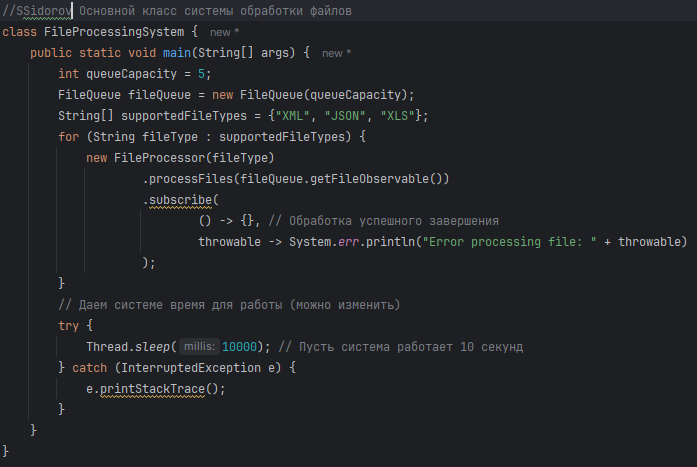


Рисунок 44 – Код основного класса

Вывод программы представлен на рисунке 45.



Рисунок 45 – Вывод программы

# Практическая работа №4

В ходе выполнения практической работы было создано клиент-серверное приложение с использованием протокола RSocket.

Был создан класс Hat – основной класс объекта данных. Он представлен на рисунке 46.

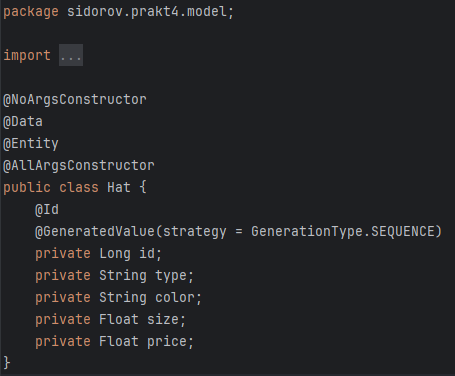


Рисунок 46 – Класс Hat

Был создан репозиторий, он представлен на рисунке 47.

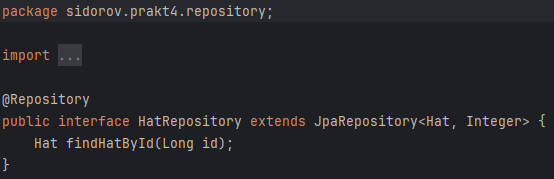


Рисунок 47 – Репозиторий HatRepository

Серверный контроллер представлен на рисунке 48.

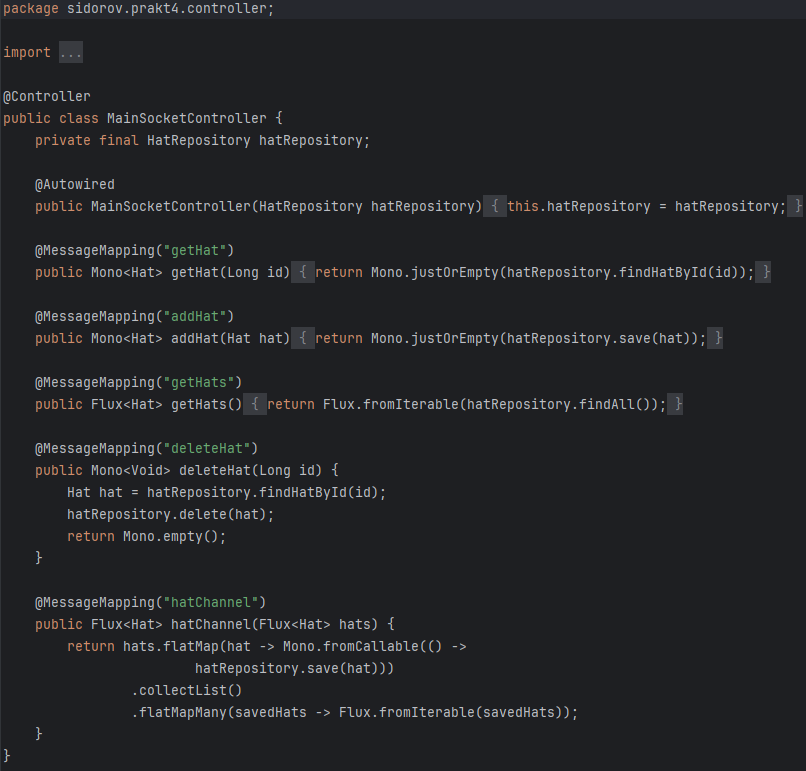


Рисунок 48 – Серверный контроллер

Конфигурация клиента для того, чтобы клиент смог связаться с

сервером через RSocket представлена на рисунке 49.



Рисунок 49 – Конфигурация клиента

Request-Response контролер представлен на рисунке 50.

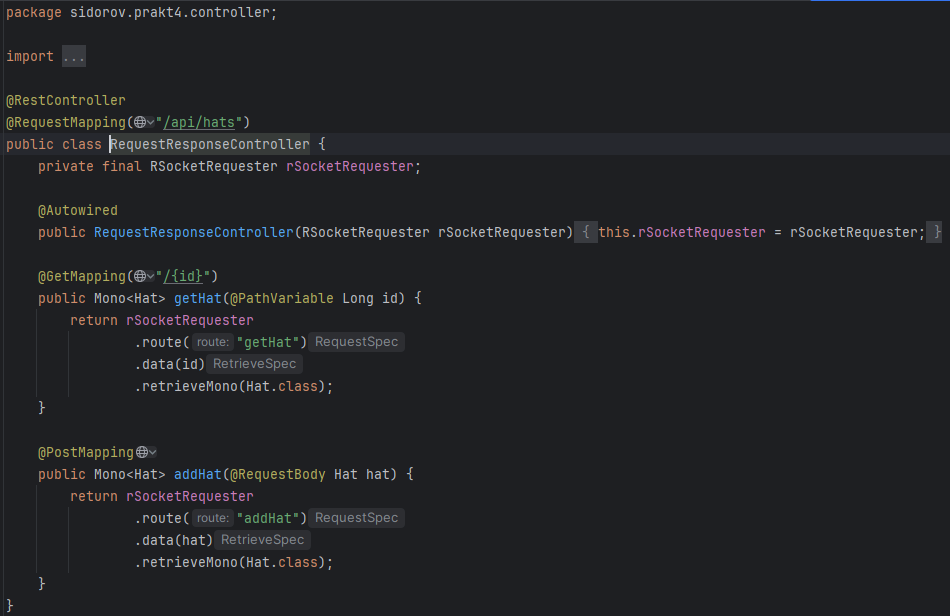


Рисунок 50 – Request-Response контроллер

Request-Stream контроллер представлен на рисунке 51.

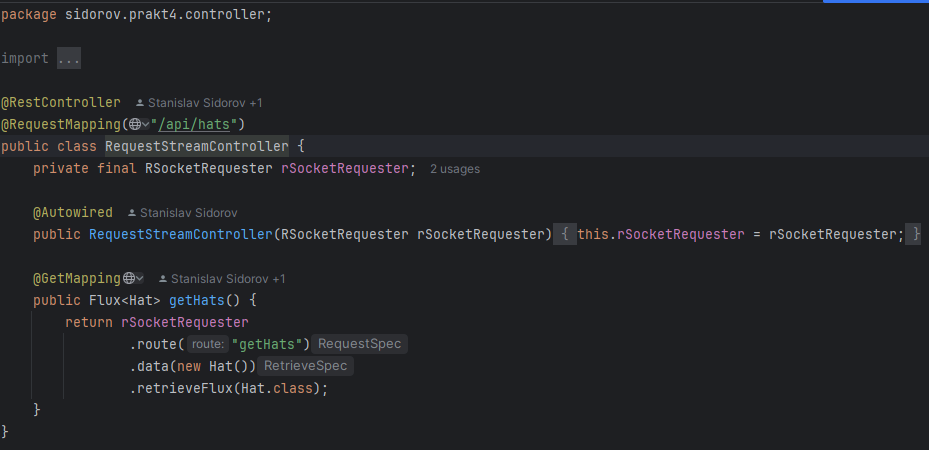


Рисунок 51 – Request-Stream контроллер

FireAndForget контроллер представлен на рисунке 52.



Рисунок 52 – FireAndForget контроллер

Channel контроллер представлен на рисунке 52.

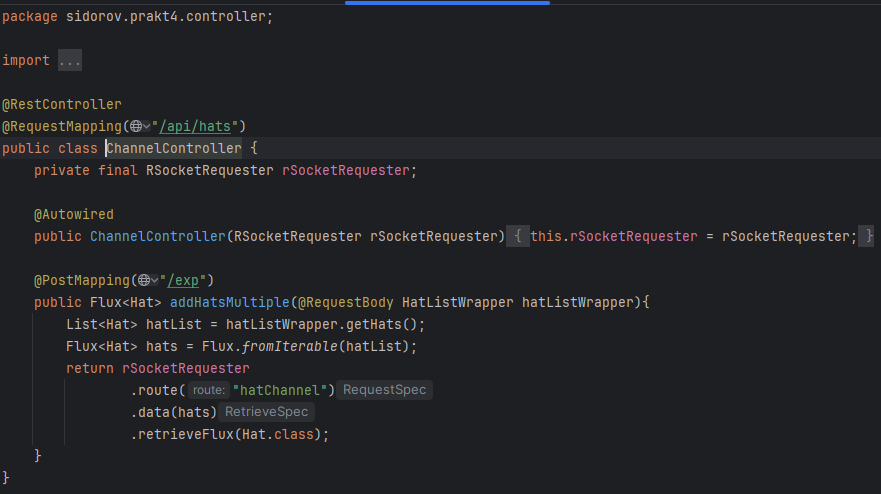


Рисунок 53 – Channel контроллер

Серверная часть была покрыта тестами. Unit тесты представлены на рисунках 54 и 55.

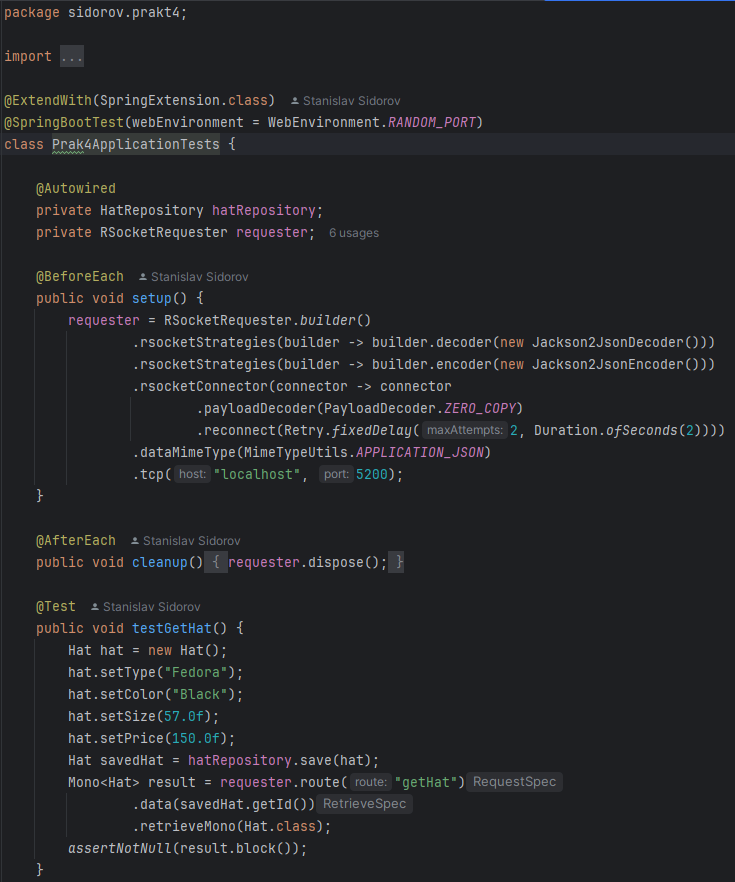


Рисунок 54 – Unit тесты



Рисунок 55 – Unit тесты

Пример работы с сервисом преставлен на рисунках 56 – 60.

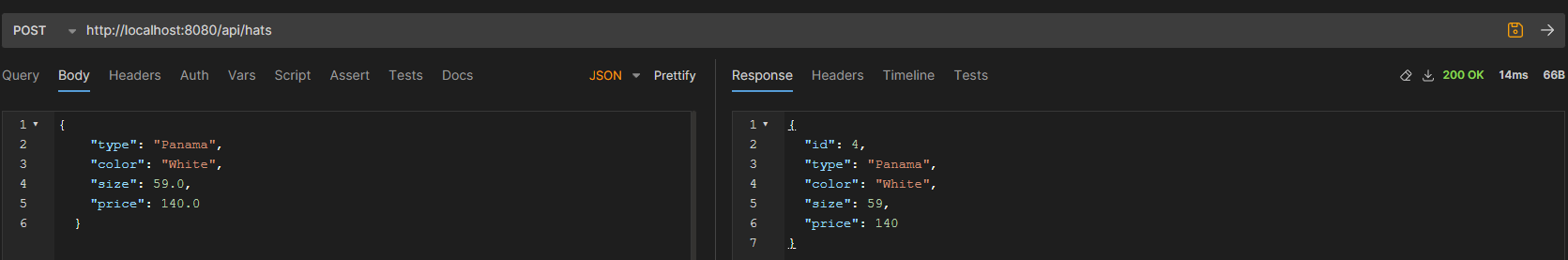
****

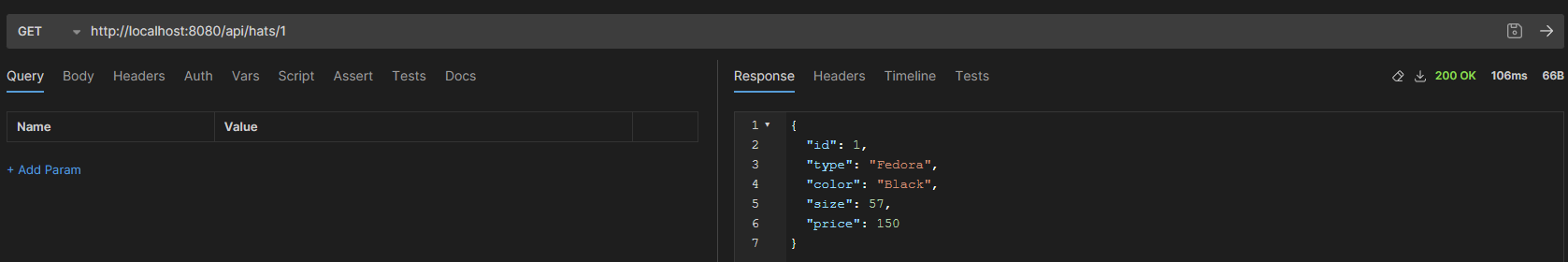
Рисунок 56 – Пример создания шляпы ****

Рисунок 57 – Пример получения шляпы

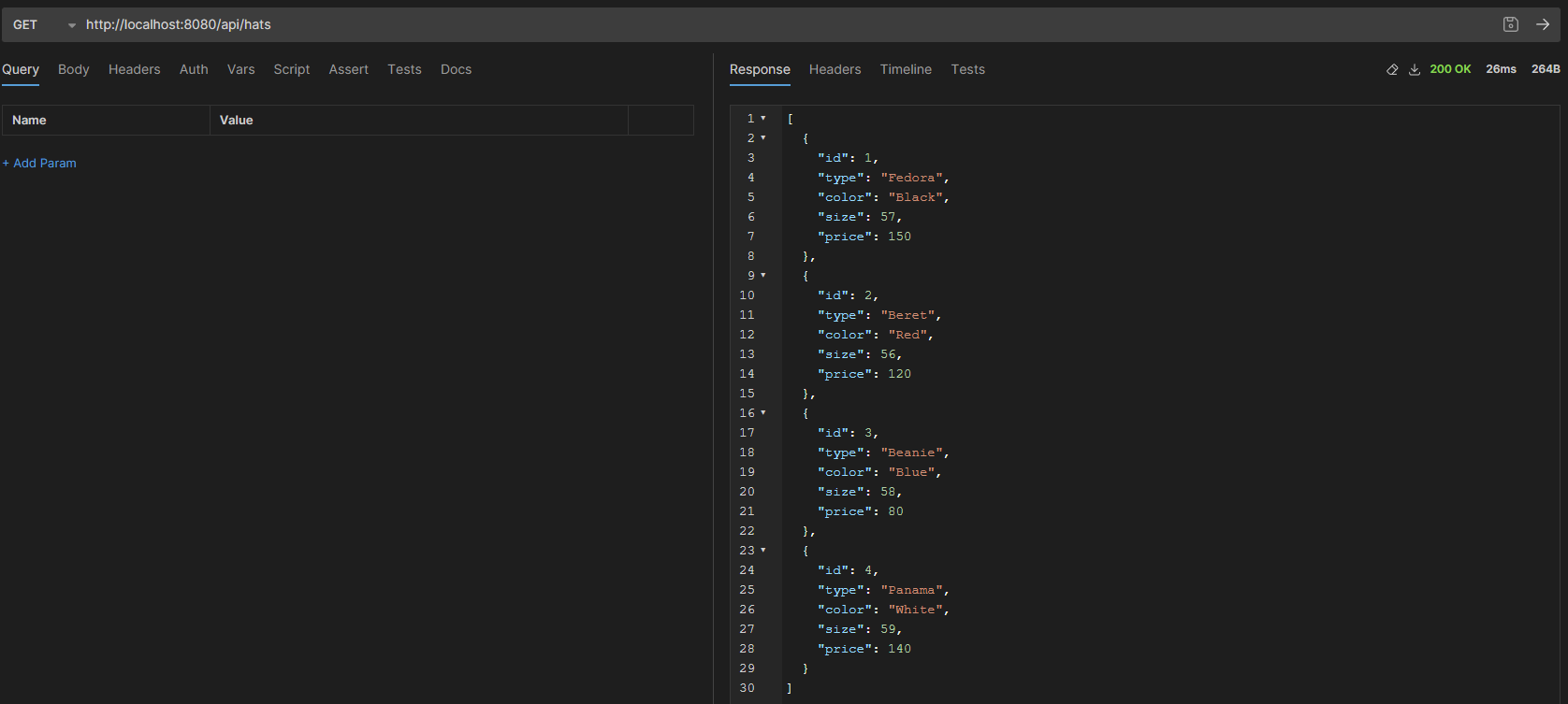
****

Рисунок 58 – Пример получения шляп

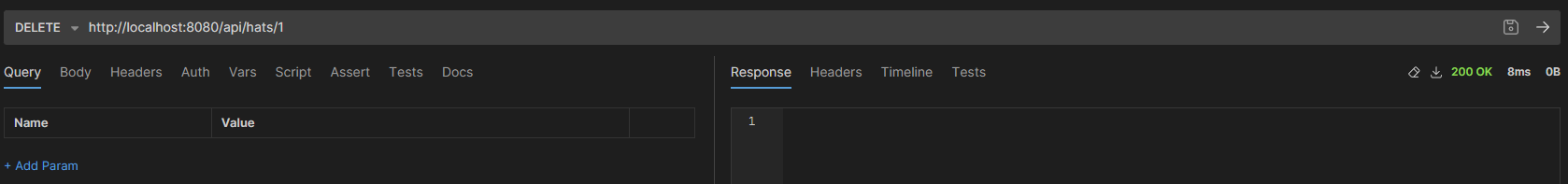
****

Рисунок 59 – Пример удаления шляпы

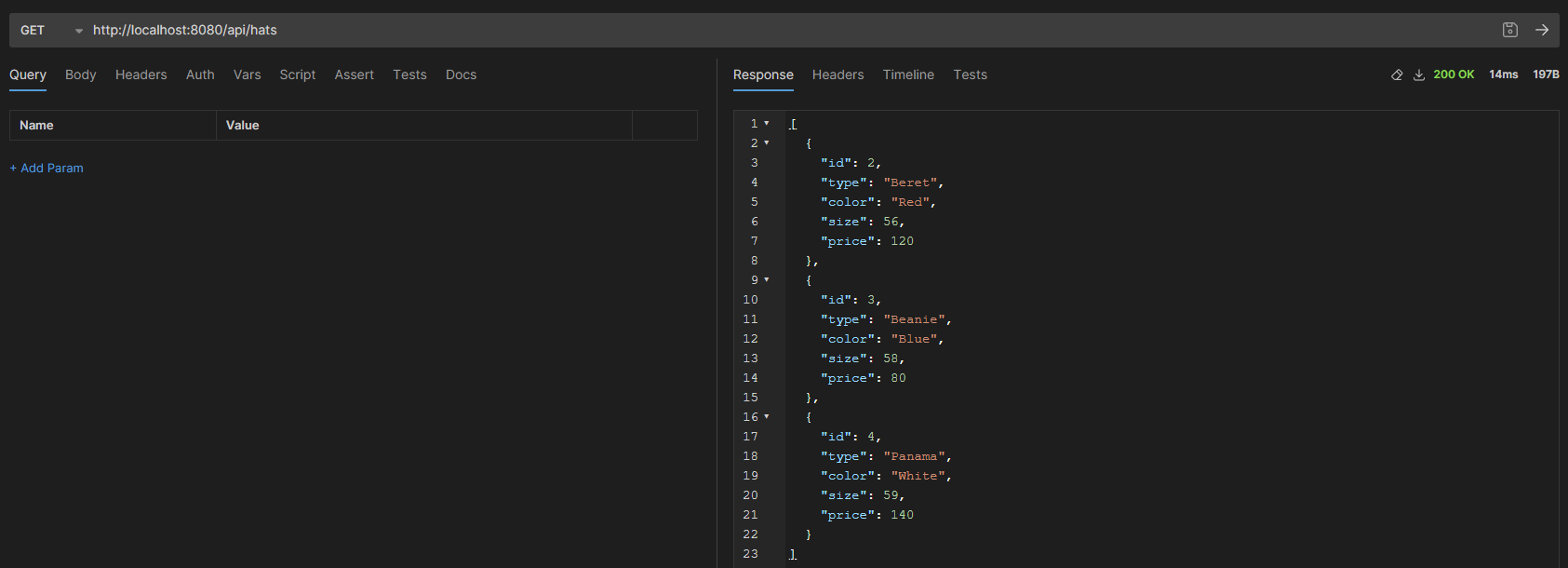
****

Рисунок 60 – Результат удаления шляпы

Взаимодействие с бд на уровне приложения представлено на рисунке 61.

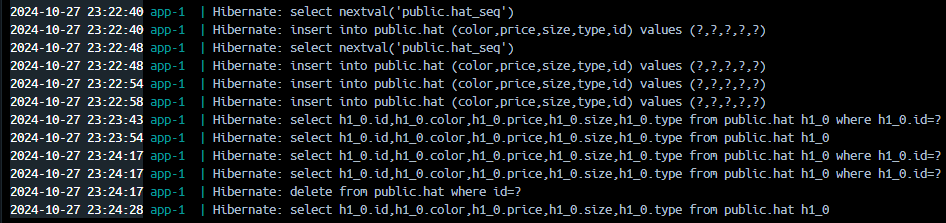
****

Рисунок 61 – Взаимодействие с бд

# Практическая работа №5

В ходе выполнения практической работы было создана децентрализованная распределенная система с использованием Spring/Spring Boot.

Было создано приложение с использованием Spring. Код приложения представлена на рисунках 62 - 65.

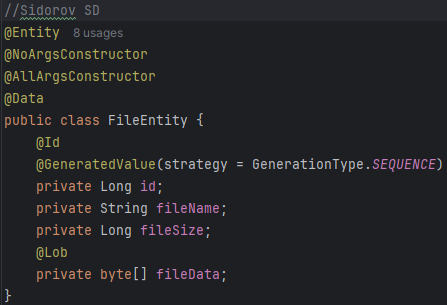


Рисунок 62 – Модель FileEntity

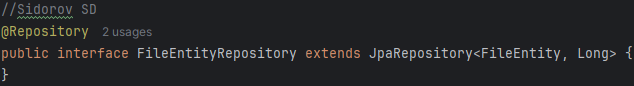


Рисунок 63 – Репозиторий

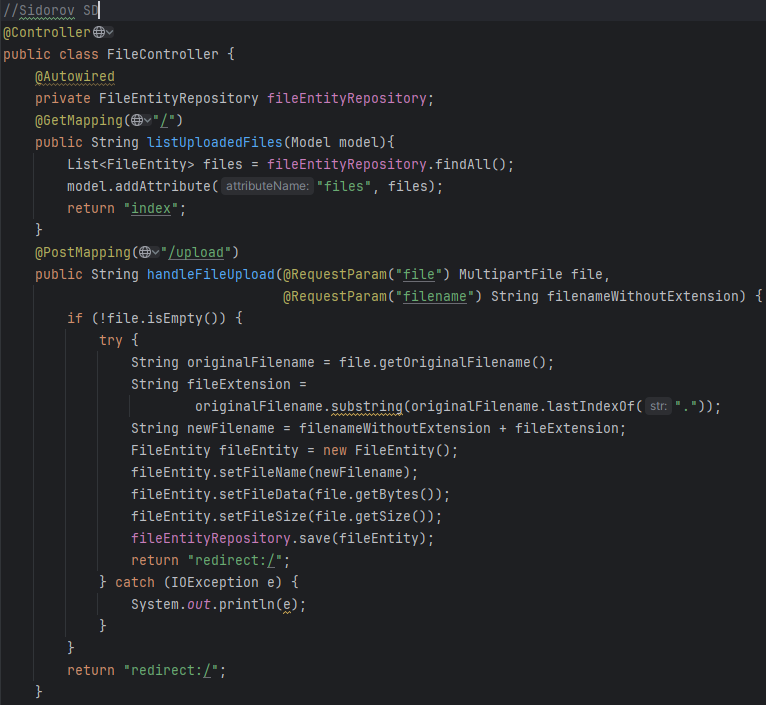


Рисунок 64 – Контроллер

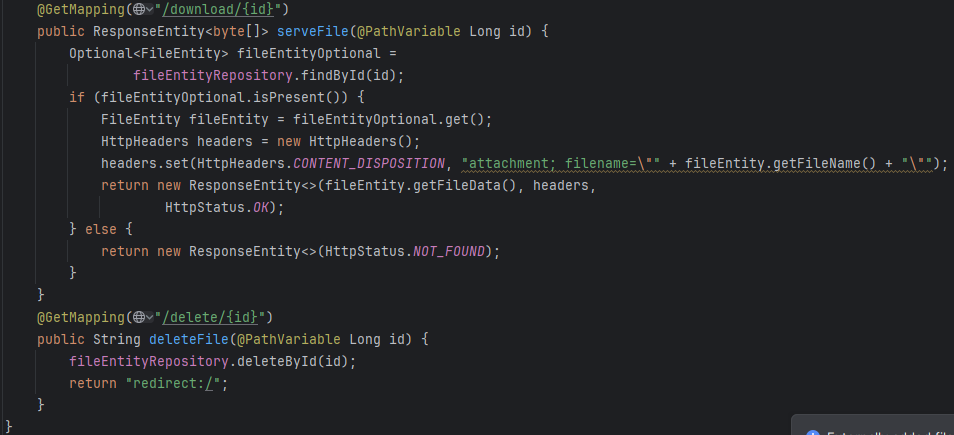


Рисунок 65 – Контроллер

Dockerfile представлен на рисунке 66.



Рисунок 66 – Dockerfile

Конфигурация nginx представлена на рисунке 67.



Рисунок 67 – nginx

Docker-compose представлен на рисункe 68.

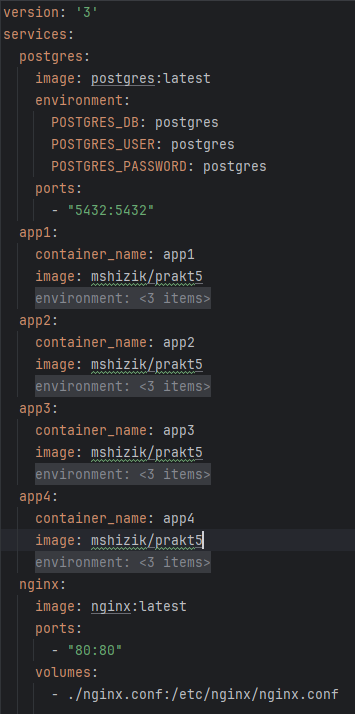


Рисунок 68 – docker-compose

Проверка работоспособности контейнеров представлена на рисунке 69.

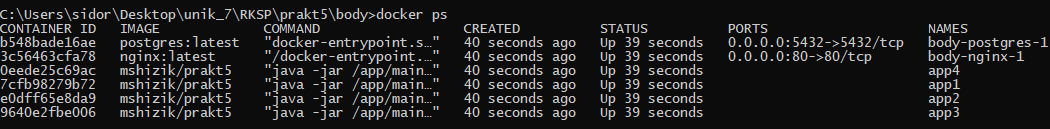


Рисунок 69 – Проверка доступности контейнеров

Поочередная остановка контейнеров и проверка работоспособности остальных представлена на рисунках 70, 71.

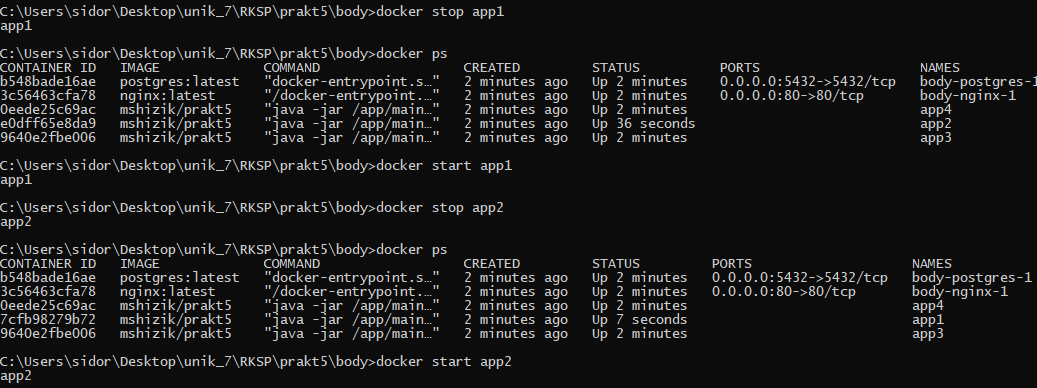


Рисунок 70 – Поочередная остановка контейнеров

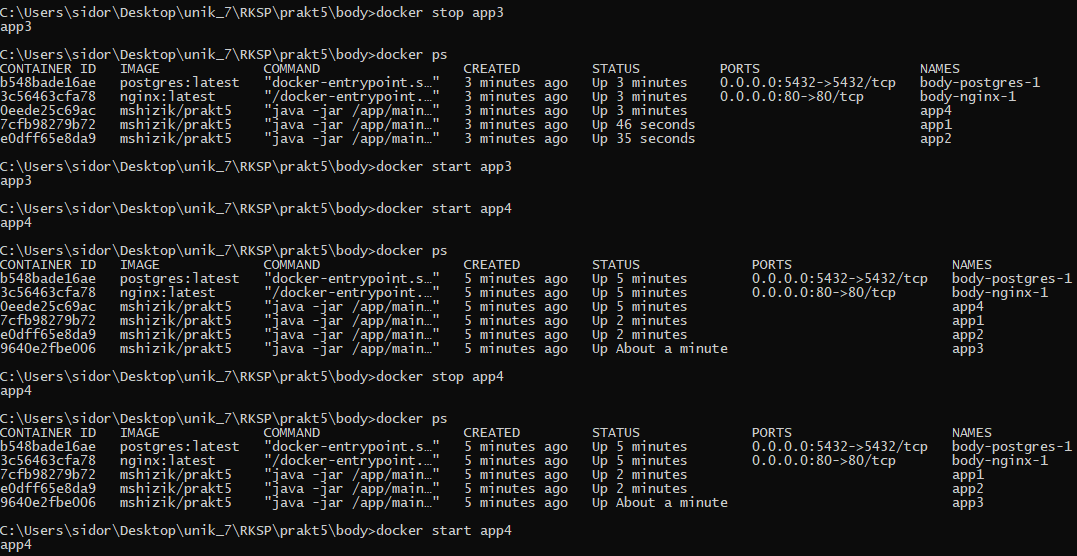


Рисунок 71 – Поочередная остановка контейнеров

Остановка двух контейнеров и проверка работоспособности остальных представлена на рисунке 72.

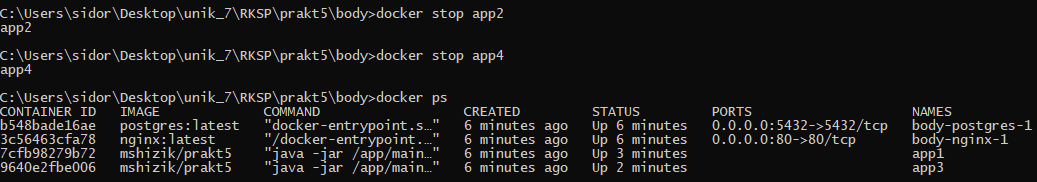


Рисунок 72 – Остановка двух контейнеров

# Практическая работа №6

Был создан смарт-контракт, моделирующий процессы покупки и обмена ценными бумаги. Код представлена на рисунке 73 и 74.



Рисунок 73 – Код

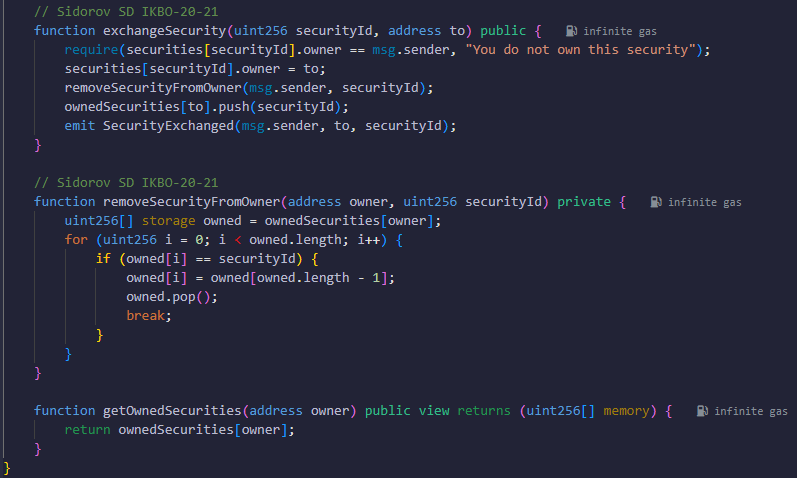


Рисунок 74 - Код

Потом были написаны тесты. Тесты представлены на рисунке 75 - 76.

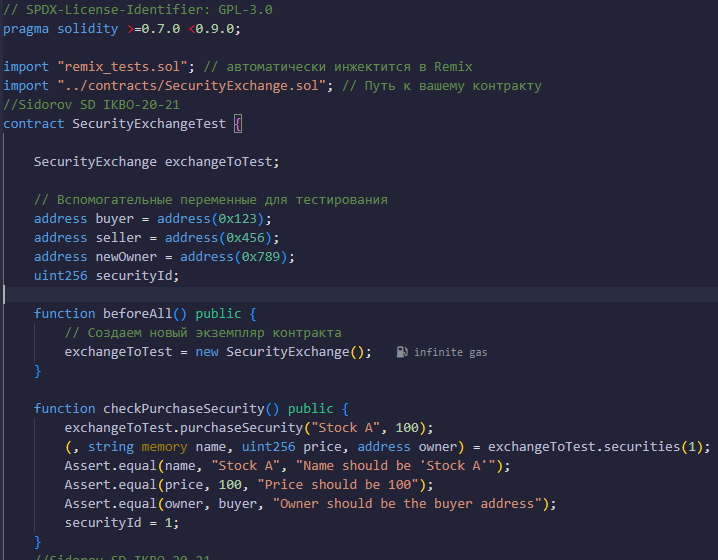


Рисунок 75 - Тесты



Рисунок 76 – Тесты

После смарт контракты и тесты были задеплоены. Деплой представлен на рисунке 77.

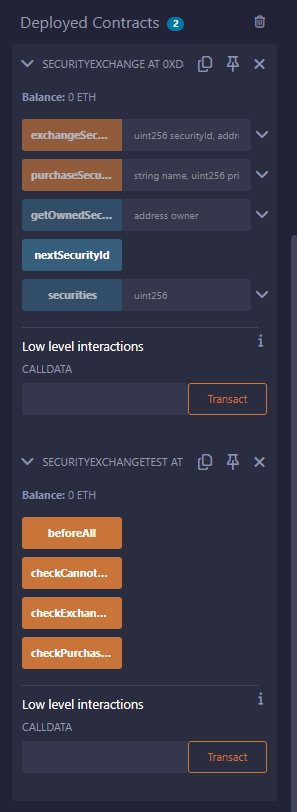


Рисунок 77 – Деплой

Проверка работоспособности представлена на рисунке 78.

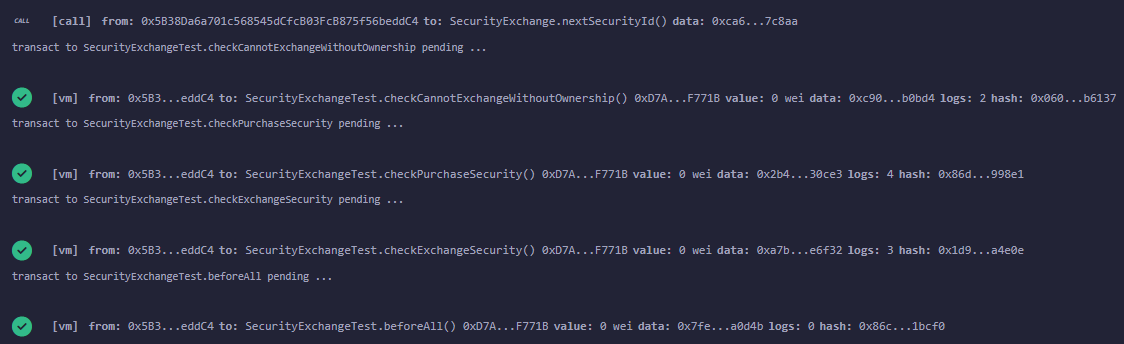


Рисунок 78 – Работоспособность

# Практическая работа №7

Успешное прохождение тестов продемонстрировано на рисунке 79.



Рисунок 79 – Успешное прохождение тестов

Использование апи продемонстрировано на рисунках 78 -83.

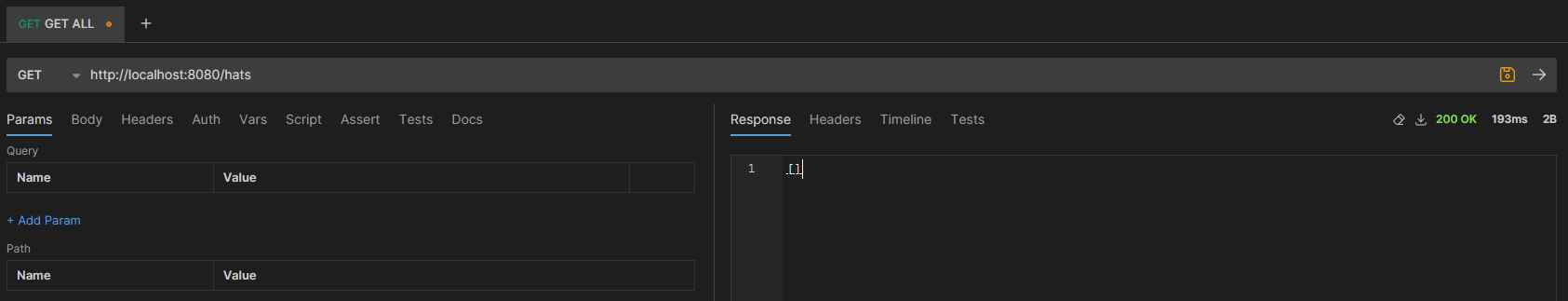


Рисунок 78 – Получение всех шляп

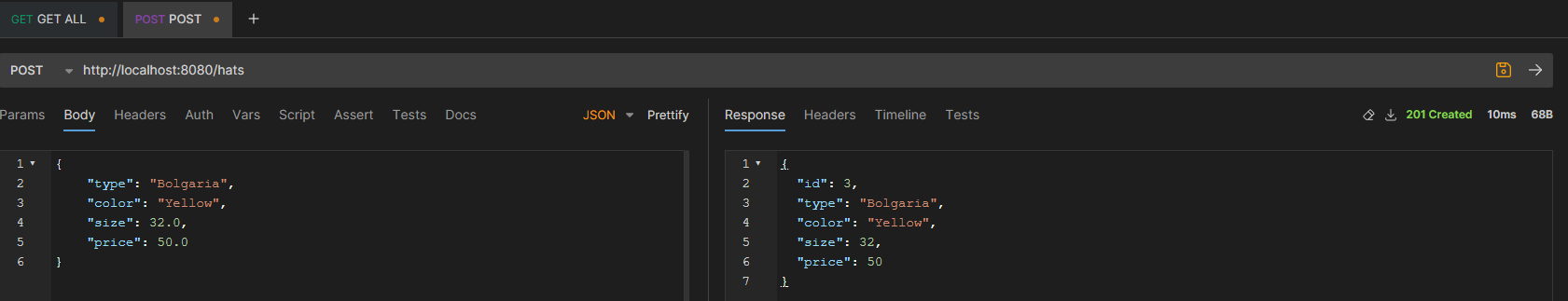


Рисунок 79 – Добавление шляпы

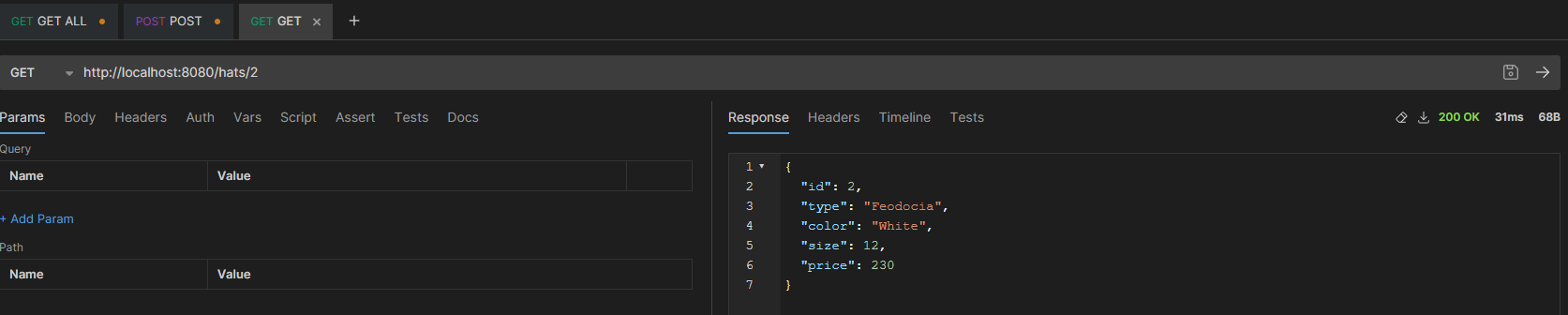


Рисунок 80 – Получение шляпы

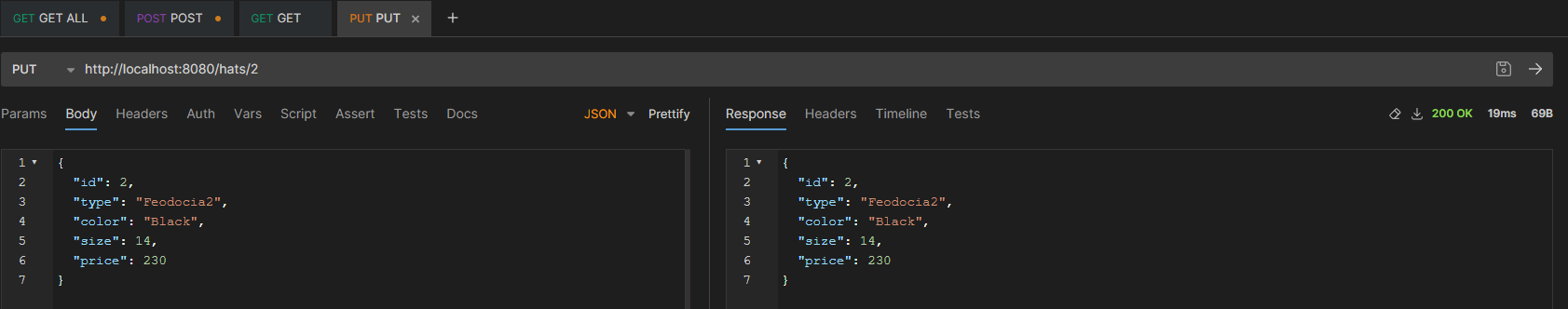


Рисунок 81 – Обновление шляпы

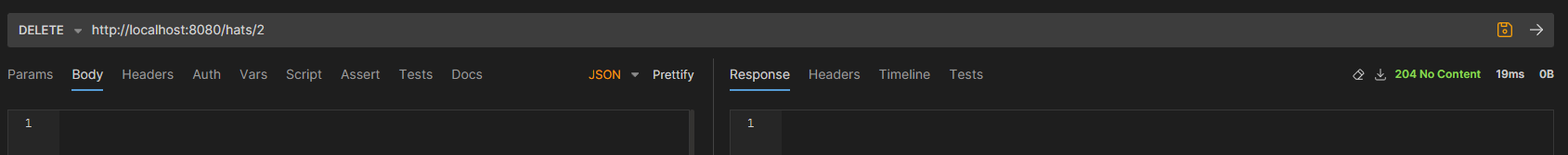


Рисунок 82 – Удаление шляпы 2

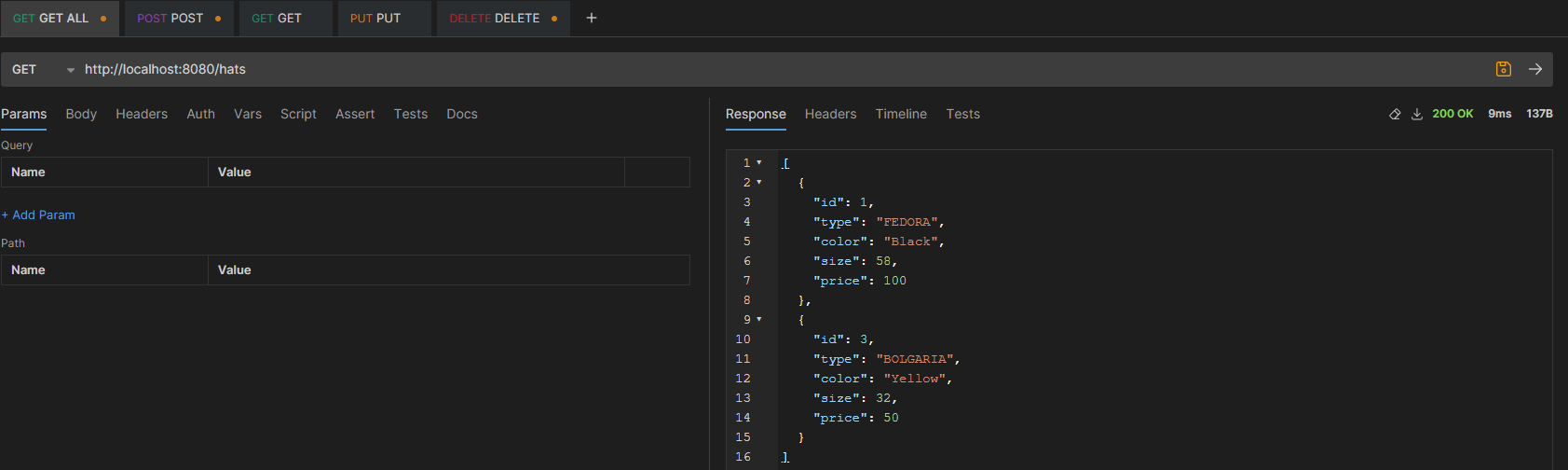
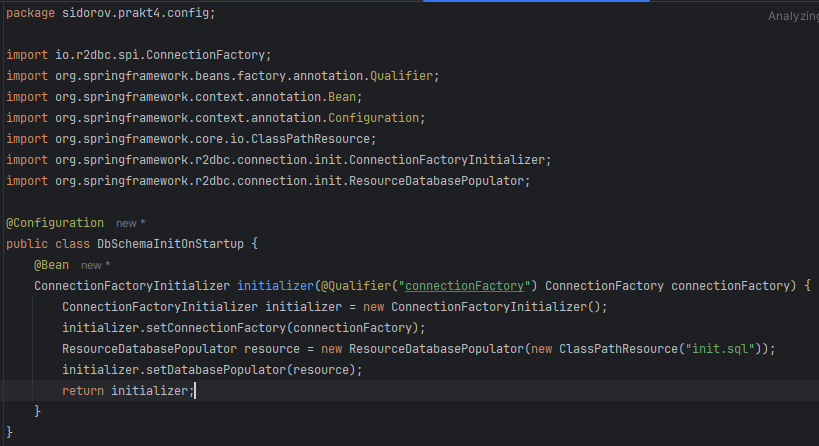
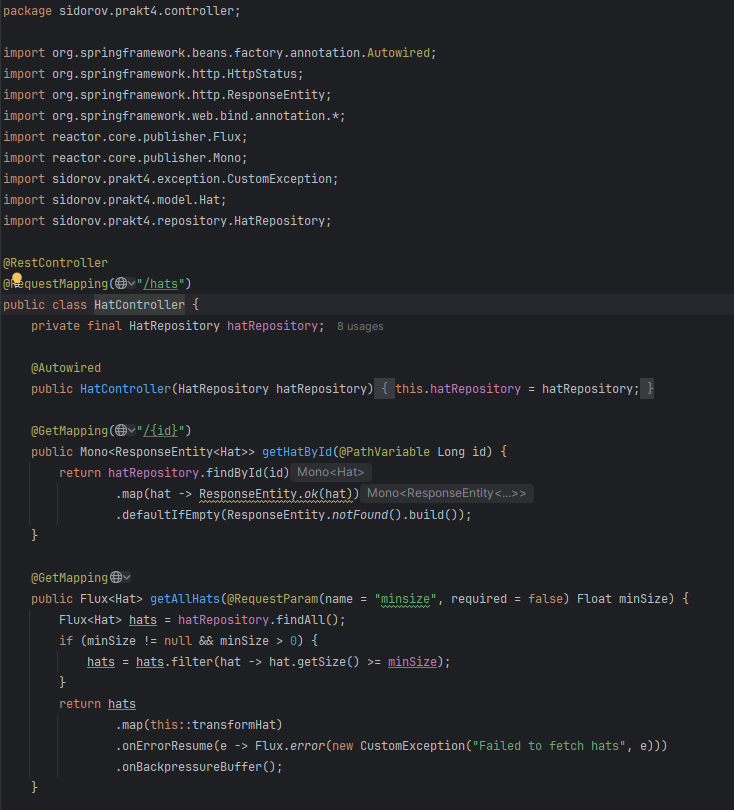
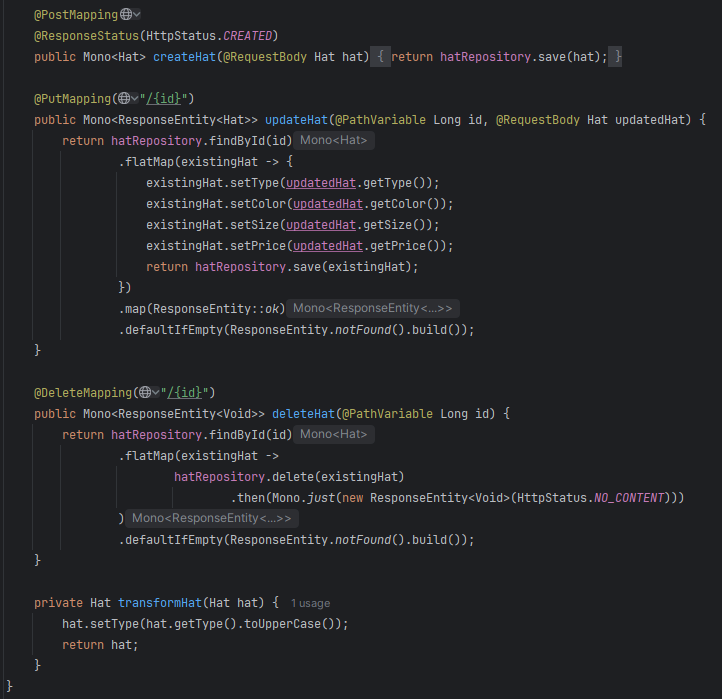


Рисунок 83 – Получение всех шляп

Используемые классы представлены на рисунках 84 – 89

Рисунок 84 – Класс для инициализации БД  
Рисунок 85 – Первая половина класса контроллера  
  
Рисунок 86 – Вторая половина класса контроллера

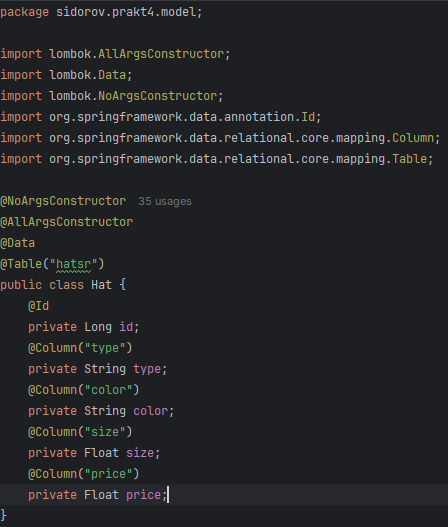


Рисунок 87 – Класс шляпы

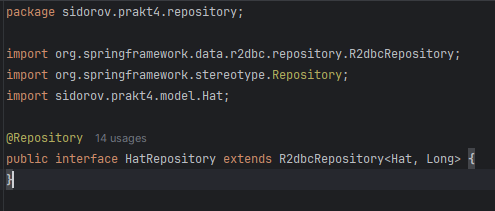
  
Рисунок 88 – Класс репозитория

  
Рисунок 89 – Конфигурация приложения

Тесты для приложения представлены на рисунках 90 - 93.



Рисунок 90 – тест

  
Рисунок 91 – тест

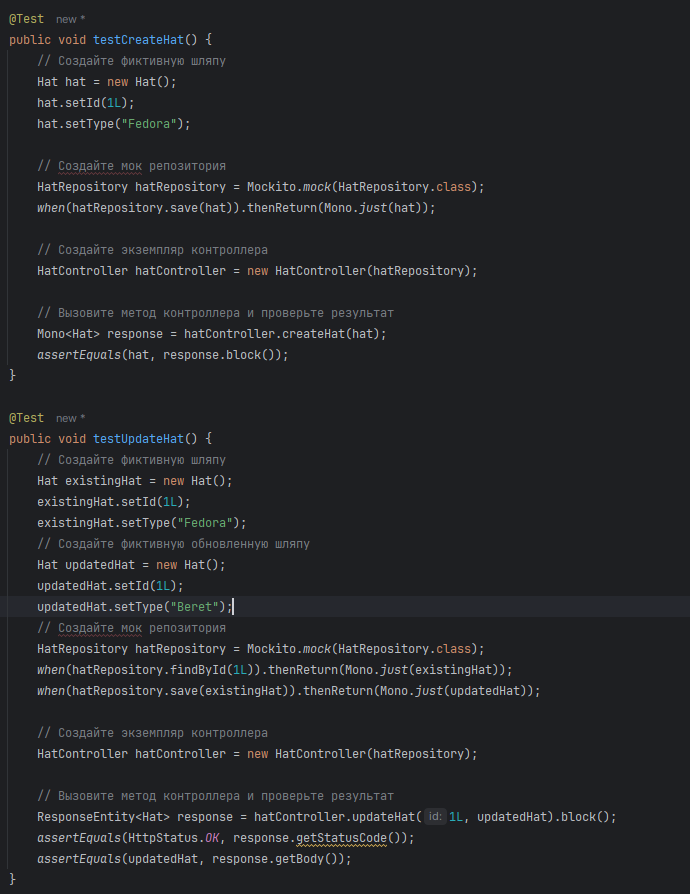


Рисунок 92 – тест

  
Рисунок 93 – тест

# Практическая работа №8

Для выполнения данной работы были выполнены шаги, описанные далее.

С помощью Spring Cloud Config Server был развернут сервер для получения конфигурации из Git. Репозиторий представлен на рисунке 94, конфигурация сервера представлена на рисунке 95.



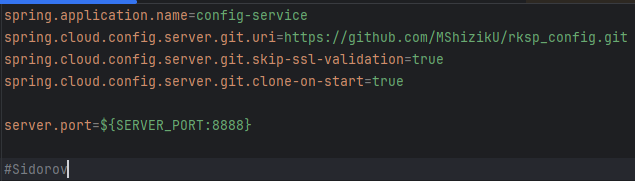
Рисунок 94 – Репозиторий с конфигурациями сервисов  


Рисунок 95 – Конфигурация сервера облачных конфигураций

Для общения сервисов между собой был развернут eureka сервер. Конфигурация сервера представлена на рисунке 96.

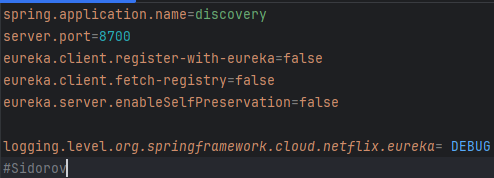


Рисунок 96 – Конфигурация сервера eureka

Далее было развернуто четыре сервиса. Сервис авторизации, сервис клиентов, сервис платежей и сервис уведомлений. Ниже будут представлены различные части каждого сервиса, демонстрирующие примерное исполнение системы.

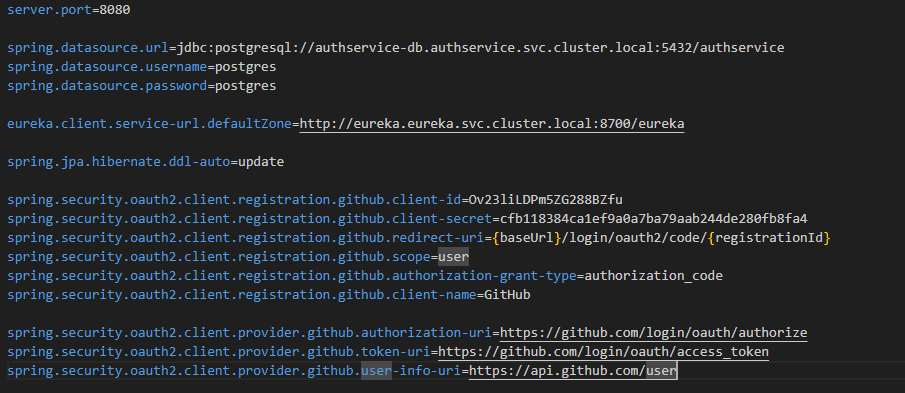


Рисунок 97 – Конфигурация сервиса авторизации

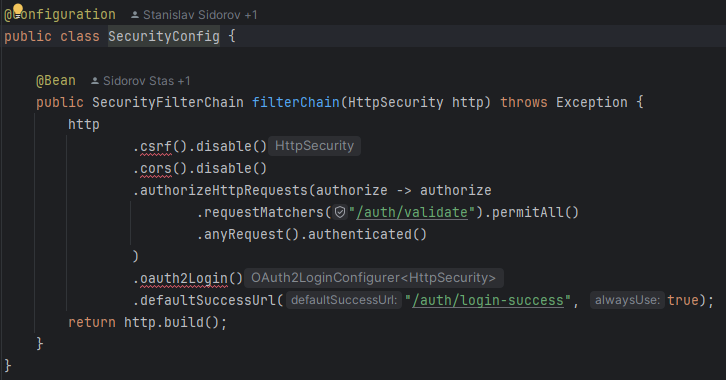


Рисунок 98 – Конфигурация безопасности сервиса авторизации

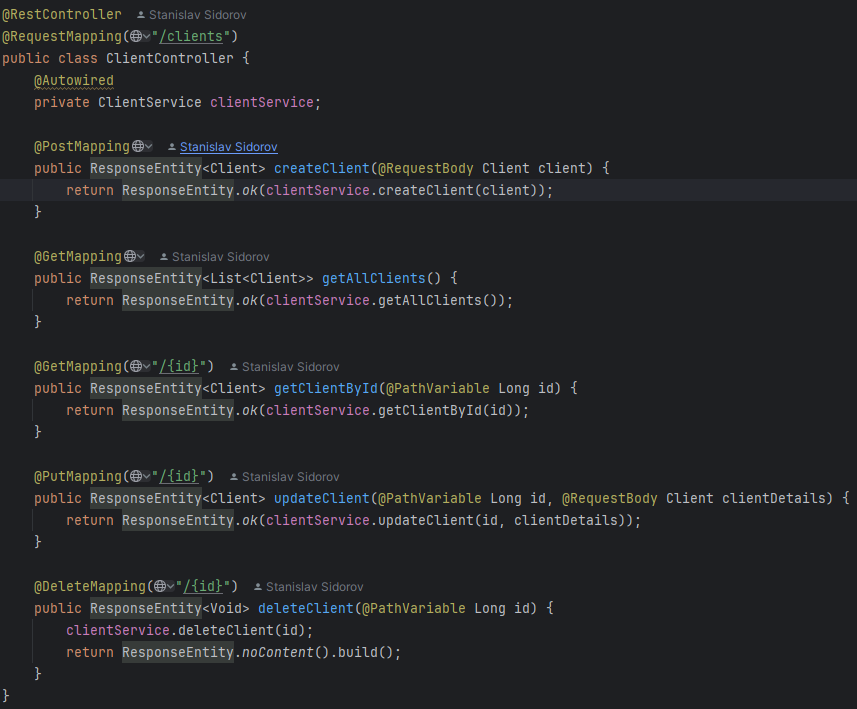


Рисунок 99 – Контроллер сервиса клиентов

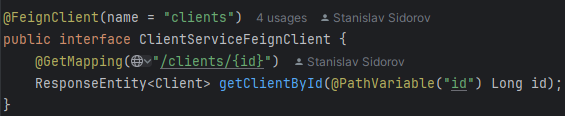


Рисунок 100 – FeignClient для получения клиента в сервисе платежей

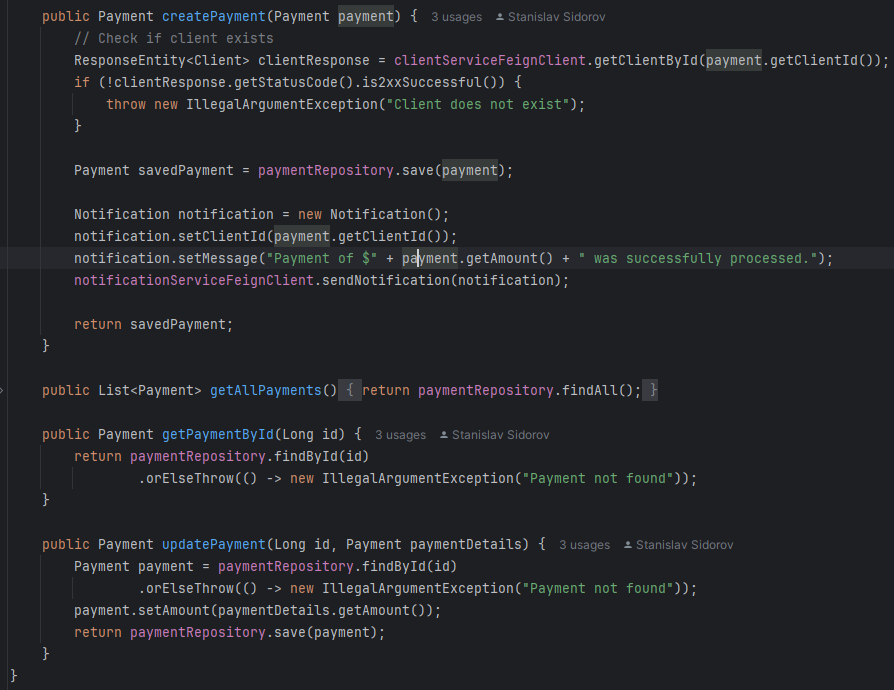


Рисунок 101 – Сервис для сервиса платежей



Рисунок 102 – Unit-тесты для сервиса уведомлений

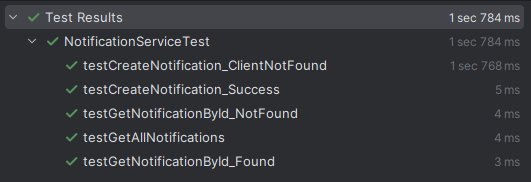


Рисунок 103 – Успешное прохождение Unit-тестов

Для реализации API Gateway был использован krakend, часть конфигурации его, представлена на рисунке 104.

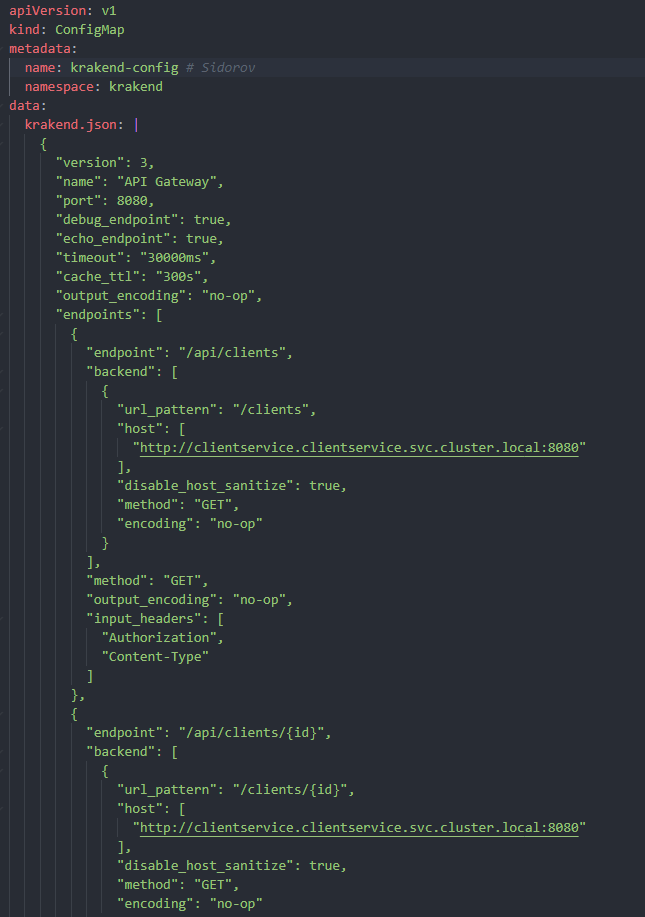


Рисунок 104 – Конфигурация krakend

Сервисы были развернуты с помощью helm чартов, а также yaml файлов, фрагменты которых представлены ниже.

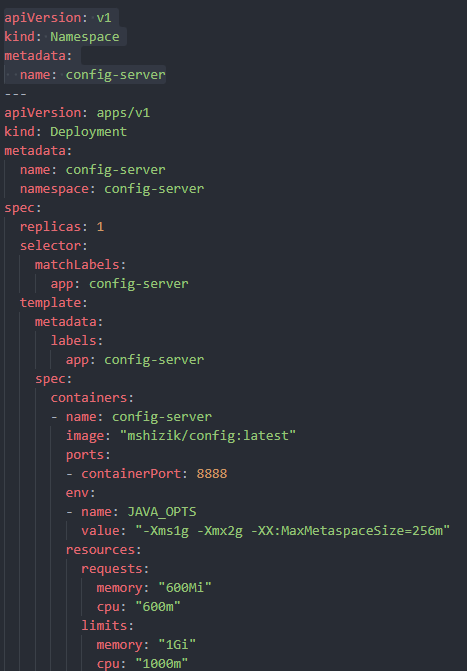


Рисунок 105 – Развертывания сервера конфигурации



Рисунок 106 – Развертывание сервера eureka

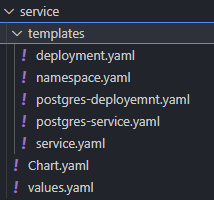


Рисунок 107 – Состав helm чарта для развертки сервисов

Все сервисы были развернуты в minikube. Получившиеся поды представлены на рисунке 108.

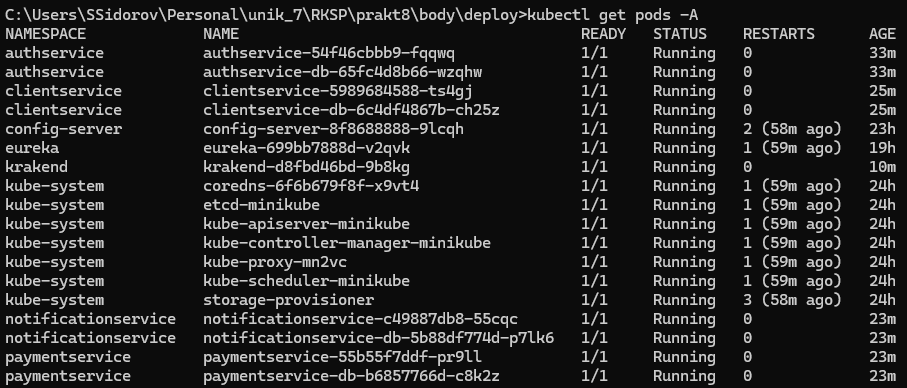


Рисунок 110 – Поды сервисов получившегося приложения

Также было проведено ручное тестирование апи, результаты которого приведены ниже на рисунках 111 - .

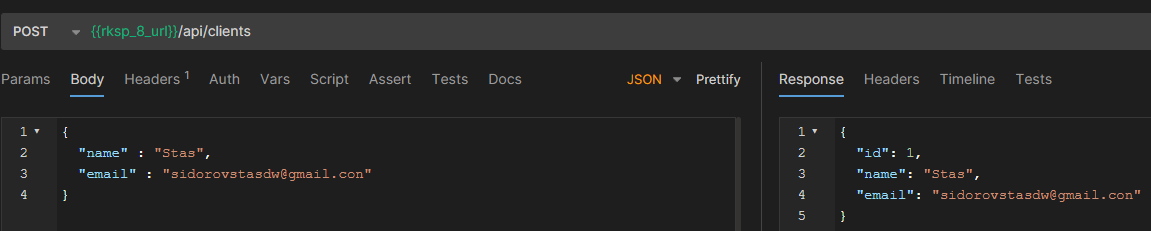


Рисунок 111 – Создание клиента

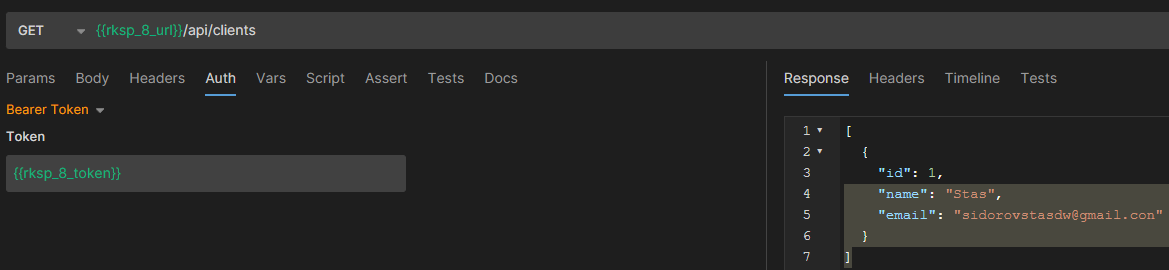


Рисунок 112 – Получение списка клиентов

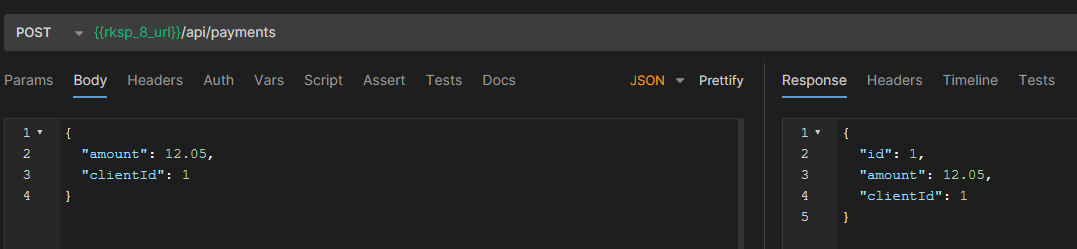


Рисунок 113 – Создание платежа

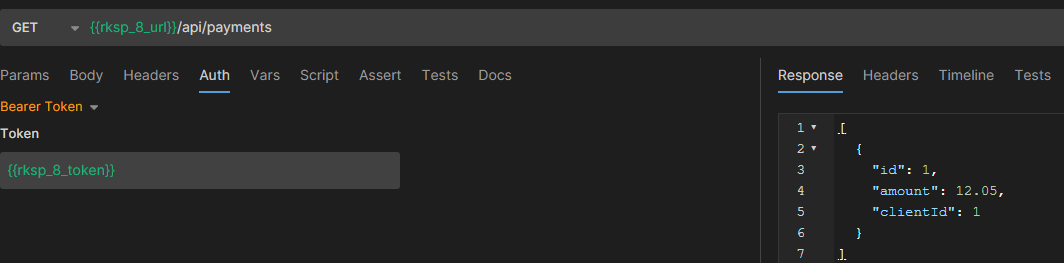


Рисунок 114 – Получение списка платежей

  
Рисунок 115 – Получение списка уведомлений о платежах