|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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 | Хитров Н.С. |
| Принял ассистент | Зарипов Е.А. |

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

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ПРАКТИЧЕСКАЯ РАБОТА №1 3](#_Toc184209220)

[ПРАКТИЧЕСКАЯ РАБОТА №2 12](#_Toc184209221)

[ПРАКТИЧЕСКАЯ РАБОТА №3 19](#_Toc184209222)

[ПРАКТИЧЕСКАЯ РАБОТА №4 30](#_Toc184209223)

[ПРАКТИЧЕСКАЯ РАБОТА №5 40](#_Toc184209224)

[ПРАКТИЧЕСКАЯ РАБОТА №6 47](#_Toc184209225)

[ПРАКТИЧЕСКАЯ РАБОТА №7 49](#_Toc184209226)

[ПРАКТИЧЕСКАЯ РАБОТА №8 57](#_Toc184209227)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 67](#_Toc184209228)

# ВВЕДЕНИЕ

Ссылка на github с практическими 1-7:   
Ссылка на github с практической 8:

# ПРАКТИЧЕСКАЯ РАБОТА №1

**Цель работы**

Развить у студентов навыки работы с многопоточностью и параллельными вычислениями в Java.

**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 за- пускекаждого из вариантов реализации.

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

1. Поиск суммы элементов массива.

2. Поиск максимального элемента в массиве.

3. Поиск минимального элемента в массиве.

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.

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 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 удовлетворять следующими условиям:

1. Должна быть обеспечена потокобезопасность.

2. Работа генератора не должна зависеть от работы обработчиков, и наоборот.

3. Если нет задач, то потоки не должны быть активны.

4. Если нет задач, то потоки не должны блокировать другие потоки.

5. Должна быть сохранена целостность данных.

**Выполнение задания**

Выполнение задания 1 отображено на листинге 1.1.

Листинг 1.1 – Task1.java

public class Concurrent {

private static long sum(List<Integer> array, int start, int end) {

long sum = 0;

for (int j = start; j < end; j++) {

sum += array.get(j);

try {

Thread.sleep(1);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

return sum;

}

public static long sum(List<Integer> array) {

int numThreads = Runtime.getRuntime().availableProcessors();

ExecutorService executor = Executors.newFixedThreadPool(numThreads);

int chunkSize = (int) Math.ceil(array.size() / (double) numThreads);

Продолжение листинга 1.1

List<Future<Long>> futures = new ArrayList<>();

for (int i = 0; i < numThreads; i++) {

int start = i \* chunkSize;

int end = Math.min(start + chunkSize, array.size());

futures.add(executor.submit(() -> sum(array, start, end)));

}

long sum = 0;

for (Future<Long> future : futures) {

try {

sum += future.get();

} catch (Exception e) {

e.printStackTrace();

}

}

executor.shutdown();

return sum;

}

}  
public class Consistent {

public static long sum(List<Integer> array) {

long sum = 0;

for (int i = 0; i < array.size(); i++) {

sum += array.get(i);

try {

Thread.sleep(1);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

return sum;

}

}

public class ForkJoinSum {

private static class SumTask extends RecursiveTask<Long> {

private final List<Integer> array;

private final int start;

private final int end;

private static final int THRESHOLD = 10;

public SumTask(List<Integer> array, int start, int end) {

this.array = array;

this.start = start;

this.end = end;

}

@Override

protected Long compute() {

if (end - start <= THRESHOLD) {

long sum = 0;

for (int i = start; i < end; i++) {

sum += array.get(i);

}

return sum;

} else {

int middle = start + (end - start) / 2;

SumTask left = new SumTask(array, start, middle);

SumTask right = new SumTask(array, middle, end);

left.fork();

Продолжение листинга 1.1

try {

Thread.currentThread().sleep(1);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

return right.compute() + left.join();

}

}

}

public static long sum(List<Integer> array) {

ForkJoinPool pool = new ForkJoinPool();

return pool.invoke(new SumTask(array, 0, array.size()));

}

}

public class Main {

public static void main(String[] args) {

Runtime runtime = Runtime.getRuntime();

runtime.gc();

List<Integer> array = new ArrayList<Integer>();

for (int i = 0; i < 10000; i++) {

array.add((int) (Math.random() \* 10000));

}

long start;

long memoryBefore;

long result;

long end;

long memoryAfter;

start = System.currentTimeMillis();

result = Consistent.sum(array);

end = System.currentTimeMillis();

memoryAfter = runtime.totalMemory() - runtime.freeMemory();

System.out.println(result);

System.out.println("Time of Consistent: " + (end - start));

System.out.println("Memory used: " + (memoryAfter));

start = System.currentTimeMillis();

result = Concurrent.sum(array);

end = System.currentTimeMillis();

memoryAfter = runtime.totalMemory() - runtime.freeMemory();

System.out.println(result);

System.out.println("Time of Concurrent: " + (end - start));

System.out.println("Memory used: " + (memoryAfter));

start = System.currentTimeMillis();

result = ForkJoinSum.sum(array);

end = System.currentTimeMillis();

memoryAfter = runtime.totalMemory() - runtime.freeMemory();

System.out.println(result);

System.out.println("Time of ForkJoin: " + (end - start));

System.out.println("Memory used: " + (memoryAfter));

}

}

На рисунке 1.1 представлен результат работы программы.

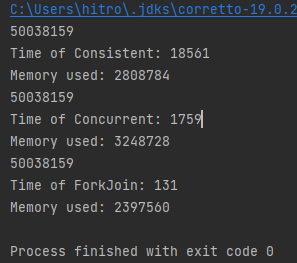


Рисунок 1.1 – Результат выполнения задания 1

Выполнение задания 2 отображено на листинге 1.2.

Листинг 1.2 – Task2.java

public class Main {

public static int square(int n) {

try {

Thread.sleep(5000);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

return n \* n;

}

public static void main(String[] args) {

ExecutorService executor = Executors.newCachedThreadPool();

Future<Integer> future = null;

Scanner scanner = new Scanner(System.in);

System.out.println("Введите число для возведения в квадрат (или 'exit' для выхода):");

while (true) {

String input = scanner.nextLine();

if ("exit".equals(input)) {

break;

}

try {

int n = Integer.parseInt(input);

future = executor.submit(() -> square(n));

System.out.println("Квадрат числа " + n + " = " + future.get());

} catch (Exception e) {

System.out.println("Некорректное значение");

}

}

executor.shutdown();

try {

if (!executor.awaitTermination(5, TimeUnit.SECONDS)) {

executor.shutdownNow();

}

} catch (InterruptedException e) {

Продолжение листинга 1.2

executor.shutdownNow();

}

scanner.close();

}

}

На рисунке 1.2 представлен результат выполнения программы.

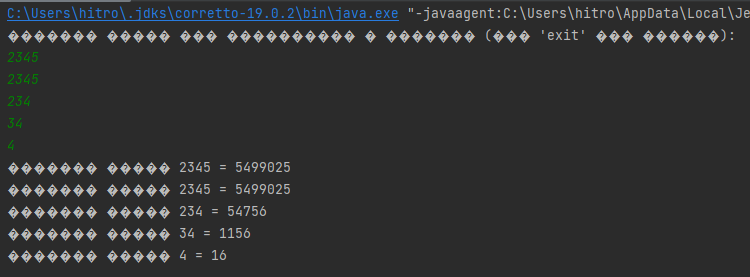


Рисунок 1.2 – Результат выполнения задания 2

Выполнение задания 3 отображено на листинге 1.3.

Листинг 1.3 – Task3.java

public class File {

private final String fileType;

private final int fileSize;

public File(String fileType, int fileSize) {

this.fileType = fileType;

this.fileSize = fileSize;

}

public String getFileType() {

return fileType;

}

public int getFileSize() {

return fileSize;

}

}

class FileGenerator implements Runnable {

private BlockingQueue<File> queue;

public FileGenerator(BlockingQueue<File> queue) {

this.queue = queue;

}

@Override

public void run() {

Random random = new Random();

String[] fileTypes = {"XML", "JSON", "XLS"};

while (true) {

try {

Thread.sleep(random.nextInt(808) + 100);

Продолжение листинга 1.3

String randomFileType = fileTypes[random.nextInt(fileTypes.length)];

int randomFileSize = random.nextInt(808) + 10;

File file = new File(randomFileType, randomFileSize);

queue.put(file); // Добавляем файл в очередь

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

break;

}

}

}

}

class FileProcessor implements Runnable {

private BlockingQueue<File> queue;

private String allowedFileType;

public FileProcessor(BlockingQueue<File> queue, String allowedFileType) {

this.queue = queue;

this.allowedFileType = allowedFileType;

}

@Override

public void run() {

while (true) {

try {

File file = queue.take(); // Получаем файл из очереди

if (file.getFileType().equals(allowedFileType)) {

long processingTime = file.getFileSize() \* 7L;

Thread.sleep(processingTime);

System.out.println("Обработан файл типа " + file.getFileType() + " с размером " + file.getFileSize() + ". Время обработки: " + processingTime + " мс.");

} else {

queue.put(file); // положили его обратно

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

break;

}

}

}

}

public class Starter {

public static void main(String[] args) {

BlockingQueue<File> queue = new LinkedBlockingQueue<>(5);

Thread generatorThread = new Thread(new FileGenerator(queue));

Thread jsonProcessorThread = new Thread(new FileProcessor(queue, "JSON"));

Thread xmlProcessorThread = new Thread(new FileProcessor(queue, "XML"));

Thread xlsProcessorThread = new Thread(new FileProcessor(queue, "XLS"));

generatorThread.start();

jsonProcessorThread.start();

xmlProcessorThread.start();

xlsProcessorThread.start();

}

}

На рисунке 1.3 представлен результат выполнения программы.

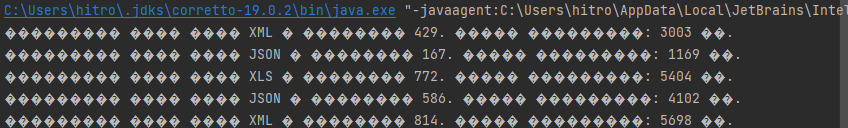


Рисунок 1.3 – Результат выполнения задания 3

**Вывод**

Использование многопоточности необходимо для ускорения обработки запросов пользователей в клиент-серверных системах, но также многопоточность не всегда позволяется ускорить выполнение задачи.

Код на github, подтверждение выполнения работы

# ПРАКТИЧЕСКАЯ РАБОТА №2

**Цель работы**

Развить у студентов практические навыки по работе с сервисным программным обеспечением и его администрированию.

**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.

**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.

**Задание 3**

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

**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(использовать реализацию из задания 3).

Если реализовать пункт 3 не представляется возможным – докажите это.

**Выполнение задания**

Выполнение задания 1 отображено на листинге 2.1.

Листинг 2.1 – Task1.java

public class task\_1 {

private static void writeLinesToFile(String fileName, List<String> lines) {

Path filePath = Paths.get(fileName);

try {

Files.write(filePath, lines);

System.out.println("Файл успешно создан: " + fileName);

} catch (IOException e) {

System.err.println("Ошибка при записи в файл: " + e.getMessage());

}

}

private static List<String> readFileContent(String fileName) {

Path filePath = Paths.get(fileName);

try {

return Files.readAllLines(filePath);

} catch (IOException e) {

System.err.println("Ошибка при чтении файла: " + e.getMessage());

}

return null;

}

public static void main(String[] args) {

String fileName = "test.txt";

List<String> fileLines = new ArrayList<>();

Random random = new Random();

int iterations = random.nextInt(5) + 5;

for (int i = 0; i < iterations; i++) {

fileLines.add(RandomStringUtils.randomAlphabetic(random.nextInt(20) + 10));

}

writeLinesToFile(fileName, fileLines);

List<String> fileContent = readFileContent(fileName);

assert fileContent != null;

for (String line : fileContent) {

System.out.println(line);

}

}

}

На рисунке 2.1 представлен результат выполнения программы.

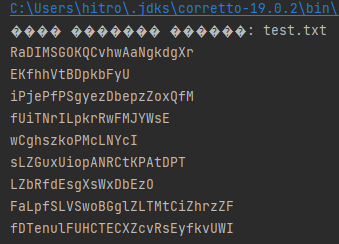


Рисунок 2.1 – Результат выполнения задания 1

Выполнение задания 2 отображено на листинге 2.2.

Листинг 2.2 – Task2.java

public class task\_2 {

public static void createFile(String name, int sizeInMB) throws IOException {

byte [] data = new byte[1024 \* 1024];

try (FileOutputStream fos = new FileOutputStream(name)) {

for (int i = 0; i < sizeInMB; i++) {

fos.write(data);

}

}

}

public static void copyFileWithStreams(String source, String dest) throws IOException {

FileInputStream fis = new FileInputStream(source);

FileOutputStream fos = new FileOutputStream(dest);

byte[] buffer = new byte[1024 \* 1024];

int bytesRead;

while ((bytesRead = fis.read(buffer)) != -1) {

fos.write(buffer, 0, bytesRead);

}

fis.close();

fos.close();

}

public static void copyFileWithChannels(String source, String dest) throws IOException {

FileInputStream fis = new FileInputStream(source);

FileOutputStream fos = new FileOutputStream(dest);

FileChannel sourceChannel = fis.getChannel();

FileChannel destinationChannel = fos.getChannel();

sourceChannel.transferTo(0, sourceChannel.size(), destinationChannel);

sourceChannel.close();

destinationChannel.close();

fis.close();

fos.close();

Продолжение листинга 2.2

}

public static void copyFileWithApacheCommonsIO(String source, String dest) throws IOException {

File sourceFile = new File(source);

File destFile = new File(dest);

FileUtils.copyFile(sourceFile, destFile);

}

private static void copyFileWithFiles(String source, String dest) throws IOException {

Path sourcePath = Path.of(source);

Path destinationPath = Path.of(dest);

Files.copy(sourcePath, destinationPath, StandardCopyOption.REPLACE\_EXISTING);

}

private static void resourceUsageReport(String method, long startTime, long endTime) {

long elapsedTime = endTime - startTime;

System.out.println("Метод " + method + ":");

System.out.println("Время выполнения: " + elapsedTime + " мс");

Runtime runtime = Runtime.getRuntime();

long memoryUsed = runtime.totalMemory() - runtime.freeMemory();

System.out.println("Использование памяти: " + memoryUsed);

System.out.println();

}

public static void main(String[] args) throws IOException {

String sourceFileName = "source.txt";

String destinationFileName = "destination.txt";

createFile(sourceFileName, 100);

// Тестирование разными методами

long startTime1 = System.currentTimeMillis();

copyFileWithStreams(sourceFileName, destinationFileName);

long endTime1 = System.currentTimeMillis();

resourceUsageReport("FileInputStream/FileOutputStream", startTime1, endTime1);

long startTime2 = System.currentTimeMillis();

copyFileWithChannels(sourceFileName, destinationFileName);

long endTime2 = System.currentTimeMillis();

resourceUsageReport("FileChannel", startTime2, endTime2);

long startTime3 = System.currentTimeMillis();

copyFileWithApacheCommonsIO(sourceFileName, destinationFileName);

long endTime3 = System.currentTimeMillis();

resourceUsageReport("Apache Commons IO", startTime3, endTime3);

long startTime4 = System.currentTimeMillis();

copyFileWithFiles(sourceFileName, destinationFileName);

long endTime4 = System.currentTimeMillis();

resourceUsageReport("Files class", startTime4, endTime4);

}

}

На рисунке 2.2 представлен результат выполнения программы.

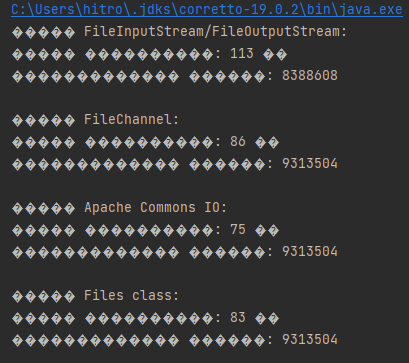


Рисунок 2.2 – Результат выполнения задания 2

Выполнение задания 3 отображено на листинге 2.3.

Листинг 2.3 – Task3.java

public class task\_3 {

public static short calculateChecksum(String filePath) throws IOException {

FileInputStream fileInputStream = new FileInputStream(filePath);

FileChannel fileChannel = fileInputStream.getChannel();

ByteBuffer buffer = ByteBuffer.allocate(2);

short checksum = 0;

while (fileChannel.read(buffer) != -1) {

buffer.flip();

while (buffer.hasRemaining()) {

checksum ^= buffer.get();

}

buffer.clear();

}

fileChannel.close();

fileInputStream.close();

return checksum;

}

public static void main(String[] args) {

String filePath = "checksum.txt";

try {

short checksum = calculateChecksum(filePath);

// %[аргумент\_индекс][флаги][ширина][.точность]спецификатор типа

// если перед шириной стоит 0, то число будет дополняться нулями

// выводим в 16-ричной

System.out.printf("Контрольная сумма файла %s: 0x%010x%n", filePath, checksum);

Продолжение листинга 2.3

|  |
| --- |
| } catch (IOException e) {  e.printStackTrace();  }  }  } |

На рисунке 2.3 представлен результат выполнения программы.



Рисунок 2.3 – Результат выполнения задания 3

Выполнение задания 4 отображено на листинге 2.4.

Листинг 2.4 – Task4.java

public class task\_4 {

private static final Map<Path, List<String>> fileContentsMap = new HashMap<>();

private static final Map<Path, String> fileHashes = new HashMap<>();

private static List<String> readLinesFromFile(Path filePath) throws IOException, FileSystemException {

List<String> lines = new ArrayList<>();

try (BufferedReader reader = Files.newBufferedReader(filePath)) {

String line;

while ((line = reader.readLine()) != null) {

lines.add(line);

}

}

return lines;

}

private static void detectFileChanges(Path filePath) throws IOException {

List<String> newFileContents = readLinesFromFile(filePath);

List<String> oldFileContents = fileContentsMap.get(filePath);

if (oldFileContents != null) {

List<String> addedLines = newFileContents.stream()

.filter(line -> !oldFileContents.contains(line))

.toList();

List<String> deletedLines = oldFileContents.stream()

.filter(line -> !newFileContents.contains(line))

.toList();

if (!addedLines.isEmpty()) {

System.out.println("added strs" + filePath + ":");

addedLines.forEach(line -> System.out.println("+ " + line));

}

if (!deletedLines.isEmpty()) {

System.out.println("del str" + filePath + ":");

deletedLines.forEach(line -> System.out.println("- " + line));

Продолжение листинга 2.4

}

} else {

System.out.println("added strs" + filePath + ":");

newFileContents.forEach(line -> System.out.println("+ " + line));

}

calculateFileHash(filePath);

fileContentsMap.put(filePath, newFileContents);

}

private static void firstObserve(Path directory) throws IOException {

try (DirectoryStream<Path> stream = Files.newDirectoryStream(directory))

{

for (Path filePath : stream) {

if (Files.isRegularFile(filePath)) {

fileContentsMap.put(filePath, readLinesFromFile(filePath));

calculateFileHash(filePath);

}

}

}

}

private static void calculateFileHash(Path filePath) {

try {

MessageDigest md = MessageDigest.getInstance("MD5");

try (InputStream is = Files.newInputStream(filePath);

DigestInputStream dis = new DigestInputStream(is, md))

{

String hash = bytesToHex(md.digest());

fileHashes.put(filePath, hash);

} catch (IOException e) {

e.printStackTrace();

}

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

}

}

private static String bytesToHex(byte[] bytes) {

StringBuilder sb = new StringBuilder();

for (byte b : bytes) {

sb.append(String.format("%02x", b));

}

return sb.toString();

}

public static void main(String[] args) throws IOException, InterruptedException {

Path directory = Paths.get("./observed");

WatchService watchService = FileSystems.getDefault().newWatchService();

directory.register(watchService,

StandardWatchEventKinds.ENTRY\_CREATE,

StandardWatchEventKinds.ENTRY\_MODIFY,

StandardWatchEventKinds.ENTRY\_DELETE);

firstObserve(directory); // предварительно обработаем все, что есть в директории

while (true) {

WatchKey key = watchService.take();

Продолжение листинга 2.4

for (WatchEvent<?> event : key.pollEvents()) {

Path filePath = (Path) event.context();

if (filePath.toString().endsWith("~")) {

continue;

}

WatchEvent.Kind<?> kind = event.kind();

if (kind == StandardWatchEventKinds.ENTRY\_CREATE) { // ивент - создание файла

System.out.println(event.context());

System.out.println("new file: " + filePath);

boolean file\_avalilable = false;

while (!file\_avalilable) {

try {

fileContentsMap.put(filePath, readLinesFromFile(directory.resolve(filePath)));

file\_avalilable = true;

} catch (FileSystemException e) {

System.out.println("Failed to obtain lock");

Thread.sleep(500);

}

}

fileContentsMap.put(filePath, readLinesFromFile(directory.resolve(filePath)));

calculateFileHash(directory.resolve(filePath));

} else if (kind == StandardWatchEventKinds.ENTRY\_MODIFY) { // ивент - изменение

System.out.println("file changed: " + filePath);

detectFileChanges(directory.resolve(filePath));

} else if (kind == StandardWatchEventKinds.ENTRY\_DELETE) { // ивент - удаление

System.out.println("file del: " + filePath);

String hash = fileHashes.get(directory.resolve(filePath));

if (hash != null) System.out.println("hash: " + hash);

}

}

key.reset();

}

}

На рисунке 2.4 представлен результат выполнения программы.

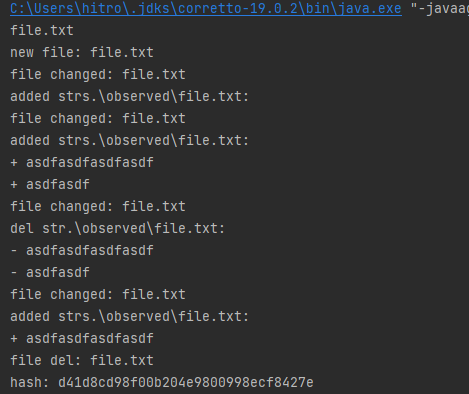


Рисунок 2.4 – Результат выполнения задания 4

**Вывод**

Ознакомились с пакетом Java NIO.

Код на github, подтверждение выполнения работы

# ПРАКТИЧЕСКАЯ РАБОТА №3

**Цель работы**

Изучение реактивного программирования и средств RxJava.

**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.

**Задание 2**

* + 1. Преобразовать поток из 1000 случайных чисел от 0 до 1000 в поток, содержащий квадраты данных чисел.
    2. 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).
    3. Дан поток из 10 случайных чисел. Сформировать поток, содержащий все числа, кроме первых трех.

**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.

**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**

Выполнение задания 1 отображено на листинге 3.1.

Листинг 3.1 – Task1.java

public class Sensors {

private static final int TEMP\_LIMIT = 25;

private static final int CO2\_LIMIT = 70;

private static final Random RANDOM\_OBJ = new Random();

public static void main(String[] args) {

CompositeDisposable publishers = new CompositeDisposable();

Observable<Integer> tempObs = Observable.interval(1, TimeUnit.SECONDS)

.map(tick -> generateTemperature())

.share();

Observable<Integer> CO2Obs = Observable.interval(1, TimeUnit.SECONDS)

.map(tick -> generateCO2())

.share();

publishers.add(

Продолжение листинга 3.1

Observable.combineLatest(

tempObs,

CO2Obs,

SensorData::new

)

.subscribeOn(Schedulers.computation())

.subscribe(sensorData -> {

boolean isTempOverLimit = sensorData.temp > TEMP\_LIMIT;

boolean isCO2OverLimit = sensorData.co2 > CO2\_LIMIT;

if (isTempOverLimit && isCO2OverLimit) {

System.out.println("ALARM - temp: " + sensorData.temp + ", CO2: " + sensorData.co2);

}

else if (isTempOverLimit) {

System.out.println("Warning - temp is over the limit: " + sensorData.temp);

}

else if (isCO2OverLimit) {

System.out.println("Warning CO2 is over the limit: " + sensorData.co2);

}

else {

System.out.println("temp: " + sensorData.temp + ", CO2: " + sensorData.co2);

}

})

);

try {

Thread.sleep(10000);

} catch (InterruptedException e) {

e.printStackTrace();

}

publishers.dispose();

}

private static int generateTemperature() {

return 15 + RANDOM\_OBJ.nextInt(16); // 15-30

}

private static int generateCO2() {

return 30 + RANDOM\_OBJ.nextInt(71); // 30-100

}

private static record SensorData(int temp, int co2) {

}

}

На рисунке 3.1 представлен результат выполнения программы.

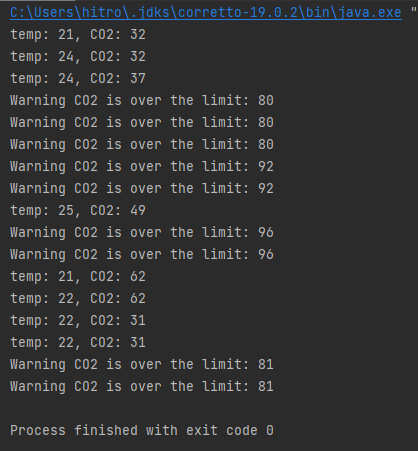


Рисунок 3.1 – Результат выполнения задания 1

Выполнение задания 2 отображено на листинге 3.2

Листинг 3.2 – Task2.java

public class First {

public static void filterNums() {

Random random = new Random();

Observable<Integer> randomNumbers = Observable.range(1, 1000)

.map(i -> random.nextInt(1001)) // 0-1000

.subscribeOn(Schedulers.computation());

Observable<Integer> filteredNums = randomNumbers

.filter(number -> number > 500);

filteredNums.subscribe(

filtered -> System.out.println(": " + filtered),

Throwable::printStackTrace,

() -> System.out.println("Фильтрация прошла успешно.")

);

try {

Thread.sleep(5000); // Достаточно времени для выполнения

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

Продолжение листинга 3.2

|  |
| --- |
| public class Second {  public static void mergeFlat() {  Random random = new Random();  // Создаем два потока случайных чисел  Observable<String> stream1 = Observable.range(1, 10)  .map(i -> "1: " + random.nextInt(10));  Observable<String> stream2 = Observable.range(1, 10)  .map(i -> "2: " + random.nextInt(10));  // Объединяем потоки  Observable<String> mergedStream = Observable.mergeDelayError(stream1, stream2);  // Подписываемся на объединенный поток и выводим элементы  mergedStream.subscribe(number -> System.out.println(number));  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  public class Third {  public static void takeFive() {  Random random = new Random();  List<Integer> list = new ArrayList<>();  Observable<Integer> randomNumbers = Observable.range(1, 10)  .map(i -> {  int num = random.nextInt(100);  list.add(num);  return num;  });  Observable<Integer> firstFiveNumbers = randomNumbers.take(5);  firstFiveNumbers.subscribe(  number -> System.out.println(number),  Throwable::printStackTrace,  () -> System.out.println(list)  );  }  } |

На рисунке 3.2 представлен результат выполнения программы.

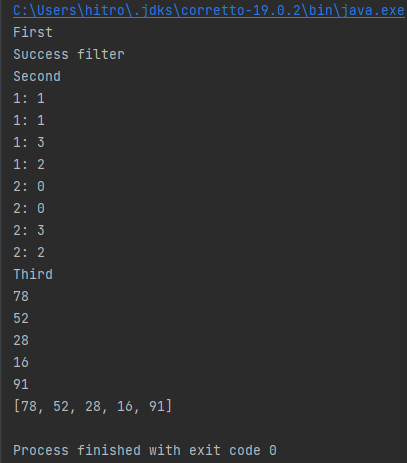


Рисунок 3.2 – Результат выполнения задания 2

Выполнение задания 3 отображено на листинге 3.3.

Листинг 3.3 – Task3.java

public class UserFriend {

private final int userId;

private final int friendId;

public UserFriend(int userId, int friendId) {

this.userId = userId;

this.friendId = friendId;

}

public int getUserId() {

return userId;

}

public int getFriendId() {

return friendId;

}

@Override

public String toString() {

return "UserFriend{" +

"userId=" + userId +

", friendId=" + friendId +

'}';

}

Продолжение листинга 3.3

}

public class Main {

private static final List<UserFriend> userFriends = new ArrayList<>();

private static final Random RANDOM = new Random();

static {

for (int i = 0; i < 1000; i++) {

int userId = 1 + RANDOM.nextInt(100);

int friendId = 1 + RANDOM.nextInt(100);

userFriends.add(new UserFriend(userId, friendId));

}

}

static Observable<UserFriend> getFriends(int userId) {

return Observable.fromIterable(userFriends)

.filter(uf -> uf.getUserId() == userId);

}

public static void main(String[] args) {

int[] userIds = new int[20];

for (int i = 0; i < userIds.length; i++) {

userIds[i] = 1 + RANDOM.nextInt(100);

}

System.out.print("User IDs: ");

for (int id : userIds) {

System.out.print(id + " ");

}

System.out.println();

Observable<Integer> userIdStream = Observable.fromArray(toIntegerArray(userIds))

.subscribeOn(Schedulers.computation());

Observable<UserFriend> friendsStream = userIdStream

.flatMap(Main::getFriends);

friendsStream.subscribe(

uf -> System.out.println("Friend: " + uf),

Throwable::printStackTrace,

() -> System.out.println("Completed processing friends.")

);

try {

Thread.sleep(5000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

private static Integer[] toIntegerArray(int[] arr) {

Integer[] result = new Integer[arr.length];

for(int i = 0; i < arr.length; i++) {

result[i] = arr[i];

}

return result;

}

}

На рисунке 3.3 представлен результат выполнения программы.

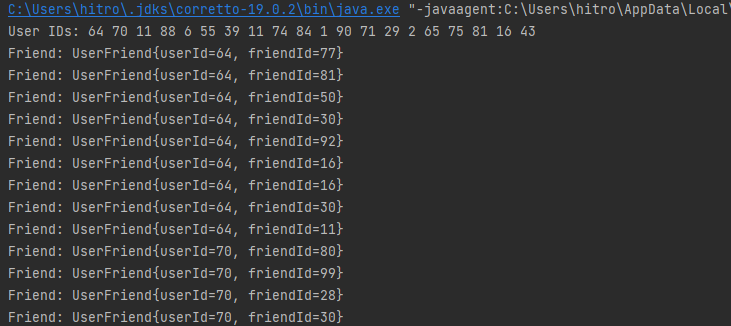


Рисунок 3.3 – Результат выполнения задания 3

Выполнение задания 4 отображено на листинге 3.4.

Листинг 3.4 – Task4.java

public class File {

private final String type;

private final int size;

public File(String type, int size) {

this.type = type;

this.size = size;

}

public String getType() {

return type;

}

public int getSize() {

return size;

}

}

public class FileGenerator {

private final Random random = new Random();

public Observable<File> generateFiles() {

return Observable.create(emitter -> {

while (!emitter.isDisposed()) {

// Генерация случайного типа файла и размера

String fileType = randomFileType();

int fileSize = random.nextInt(91) + 10; // Размер от 10 до 100

// Создание файла

File file = new File(fileType, fileSize);

emitter.onNext(file);

// Задержка от 100 до 1000 мс

TimeUnit.MILLISECONDS.sleep(random.nextInt(901) + 100);

}

Продолжение листинга 3.4

});

}

private String randomFileType() {

String[] types = {"XML", "JSON", "XLS"};

return types[random.nextInt(types.length)];

}

}

public class FileHandler {

private final String type;

public FileHandler(String type) {

this.type = type;

}

public void processFile(File file) {

if (file.getType().equals(type)) {

int processingTime = file.getSize() \* 7; // Время обработки в миллисекундах

System.out.println("Обработка файла " + file.getType() + " размером " + file.getSize() + "...");

try {

Thread.sleep(processingTime);

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

System.out.println("Файл " + file.getType() + " обработан.");

} else {

System.out.println("Обработчик не может обработать файл типа " + file.getType());

}

}

}

public class FileQueue {

private final BlockingQueue<File> queue; // Блокирующая очередь

private final int capacity;

public FileQueue(int capacity) {

this.capacity = capacity;

this.queue = new LinkedBlockingQueue<>(capacity); // Устанавливаем вместимость

}

// Добавление файла в очередь

public void enqueue(File file) {

try {

queue.put(file); // Блокирует, если очередь полная

System.out.println("Файл добавлен в очередь");

} catch (InterruptedException e) {

System.err.println("Ошибка при добавлении файла в очередь: " + e.getMessage());

Thread.currentThread().interrupt();

}

}

// Извлечение файлов из очереди в виде Observable

public Observable<File> getFiles() {

return Observable.create(emitter -> {

try {

while (!emitter.isDisposed()) {

Продолжение листинга 3.4

File file = queue.take(); // Блокирует, если очередь пуста

System.out.println("Файл извлечен из очереди");

emitter.onNext(file); // Отправляем файл в Observable

}

} catch (InterruptedException e) {

emitter.onError(e); // Сообщаем об ошибке

Thread.currentThread().interrupt(); // Восстанавливаем статус прерывания

}

});

}

}

public class Main {

public static void main(String[] args) {

FileGenerator generator = new FileGenerator();

FileQueue queue = new FileQueue(5); // Вместимость очереди 5

FileHandler jsonHandler = new FileHandler("JSON");

FileHandler xmlHandler = new FileHandler("XML");

FileHandler xlsHandler = new FileHandler("XLS");

// Генерация файлов в отдельном потоке

Observable<File> fileObservable = generator.generateFiles()

.subscribeOn(Schedulers.io())

.doOnNext(queue::enqueue);

// Обработка файлов из очереди

queue.getFiles()

.subscribeOn(Schedulers.io())

.observeOn(Schedulers.computation())

.subscribe(file -> {

jsonHandler.processFile(file);

xmlHandler.processFile(file);

xlsHandler.processFile(file);

});

// Запускаем генерацию файлов

fileObservable.subscribe();

// Добавляем задержку, чтобы программа не завершилась сразу

try {

Thread.sleep(10000); // Задержка 10 секунд

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

На рисунке 3.4 представлен результат выполнения программы.

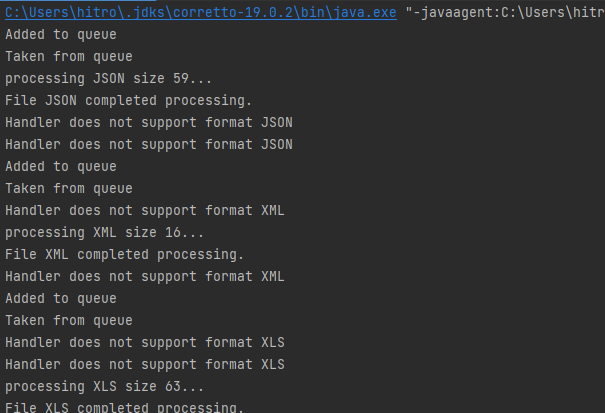


Рисунок 3.4 – Результат выполнения задания 4

**Вывод**

Изучили принципы реактивного программирования и опробовали средства RxJava для реактивного программирования.

Код на github, подтверждение выполнения работы

# ПРАКТИЧЕСКАЯ РАБОТА №4

**Цель работы**

Развить у студентов навыки проектирования и разработки клиент-серверных систем с использованием современного протокола RSocket. В ходе выполнения задания студенты освоят основные типы взаимодействия в RSocket, работу с потоками данных, интеграцию с базой данных, а также основы тестирования серверных приложений.

**Задание на практическую работу**

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 рамках протокола RSocket:

1. Request-Response;

2. Request-Stream;

3. Channel;

4. 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**

Листинг 4.1 – CatController.java

|  |
| --- |
| package com.example.pract4.controllers;  import com.example.pract4.models.Cat; import com.example.pract4.repositories.CatRepository; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.messaging.handler.annotation.MessageMapping; import org.springframework.stereotype.Controller; import reactor.core.publisher.Flux; import reactor.core.publisher.Mono;  @Controller public class CatController {  private final CatRepository catRepository;  @Autowired  public CatController(CatRepository catRepository) {  this.catRepository = catRepository;  }  @MessageMapping("getCat")  public Mono<Cat> getCat(Long id) {  return Mono.*justOrEmpty*(catRepository.findCatById(id));  }  @MessageMapping("addCat")  public Mono<Cat> addCat(Cat cat) {  return Mono.*justOrEmpty*(catRepository.save(cat));  }  @MessageMapping("getCats")  public Flux<Cat> getCats() {  return Flux.*fromIterable*(catRepository.findAll());  }  @MessageMapping("deleteCat")  public Mono<Void> deleteCat(Long id){  Cat cat = catRepository.findCatById(id);  catRepository.delete(cat);  return Mono.*empty*();  }  @MessageMapping("catChannel")  public Flux<Cat> catChannel(Flux<Cat> cats){  return cats.flatMap(cat -> Mono.*fromCallable*(() -> catRepository.save(cat)))  .collectList()  .flatMapMany(savedCats -> Flux.*fromIterable*(savedCats));  } } |

Листинг 4.2 – RequestResponseController.java

|  |
| --- |
| package com.example.pract4client.controllers;  import com.example.pract4client.models.CatModel; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.messaging.rsocket.RSocketRequester; |

Продолжение листинга 4.2

|  |
| --- |
| import org.springframework.web.bind.annotation.\*; import reactor.core.publisher.Mono;  @RestController @RequestMapping("/api/cats") public class RequestResponseController {  private final RSocketRequester rSocketRequester;  @Autowired  public RequestResponseController(RSocketRequester rSocketRequester) {  this.rSocketRequester = rSocketRequester;  }  @GetMapping("/{id}")  public Mono<CatModel> getCat(@PathVariable Long id) {  return rSocketRequester  .route("getCat")  .data(id)  .retrieveMono(CatModel.class);  }  @PostMapping  public Mono<CatModel> addCat(@RequestBody CatModel cat) {  return rSocketRequester  .route("addCat")  .data(cat)  .retrieveMono(CatModel.class);  } } |

Листинг 4.3 – ChannelController.java

|  |
| --- |
| package com.example.pract4client.controllers;  import com.example.pract4client.DTOs.CatListWrapper; import com.example.pract4client.models.CatModel; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.messaging.rsocket.RSocketRequester; import org.springframework.web.bind.annotation.PostMapping; import org.springframework.web.bind.annotation.RequestBody; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RestController; import reactor.core.publisher.Flux;  import java.util.List;  @RestController @RequestMapping("/api/cats") public class ChannelController {  private final RSocketRequester rSocketRequester;  @Autowired  public ChannelController(RSocketRequester rSocketRequester) {  this.rSocketRequester = rSocketRequester;  }  @PostMapping("/exp")  public Flux<CatModel> addCatsMultiple(@RequestBody CatListWrapper |

Продолжение листинга 4.2

|  |
| --- |
| catListWrapper){  List<CatModel> catList = catListWrapper.getCats();  Flux<CatModel> cats = Flux.*fromIterable*(catList);  return rSocketRequester  .route("catChannel")  .data(cats)  .retrieveFlux(CatModel.class);  } } |

Листинг 4.4 – ChannelController.java

|  |
| --- |
| package com.example.pract4client.controllers;  import com.example.pract4client.DTOs.CatListWrapper; import com.example.pract4client.models.CatModel; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.messaging.rsocket.RSocketRequester; import org.springframework.web.bind.annotation.PostMapping; import org.springframework.web.bind.annotation.RequestBody; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RestController; import reactor.core.publisher.Flux;  import java.util.List;  @RestController @RequestMapping("/api/cats") public class ChannelController {  private final RSocketRequester rSocketRequester;  @Autowired  public ChannelController(RSocketRequester rSocketRequester) {  this.rSocketRequester = rSocketRequester;  }  @PostMapping("/exp")  public Flux<CatModel> addCatsMultiple(@RequestBody CatListWrapper catListWrapper){  List<CatModel> catList = catListWrapper.getCats();  Flux<CatModel> cats = Flux.*fromIterable*(catList);  return rSocketRequester  .route("catChannel")  .data(cats) |

Ниже будет представлено тестирование с помощью Postman (Рисунки 4.1 – 4.4)

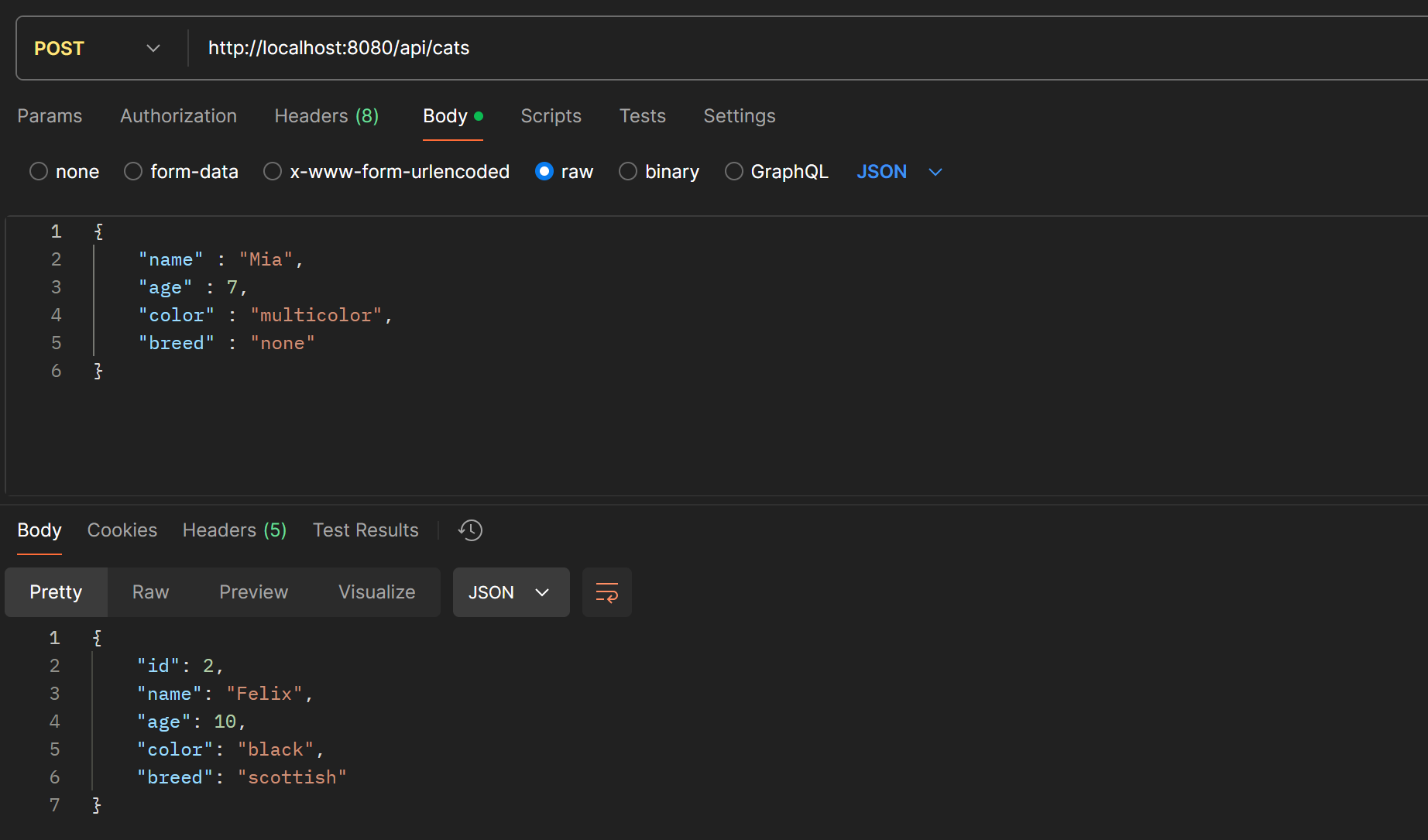


Рисунок 4.1 – Post-запрос

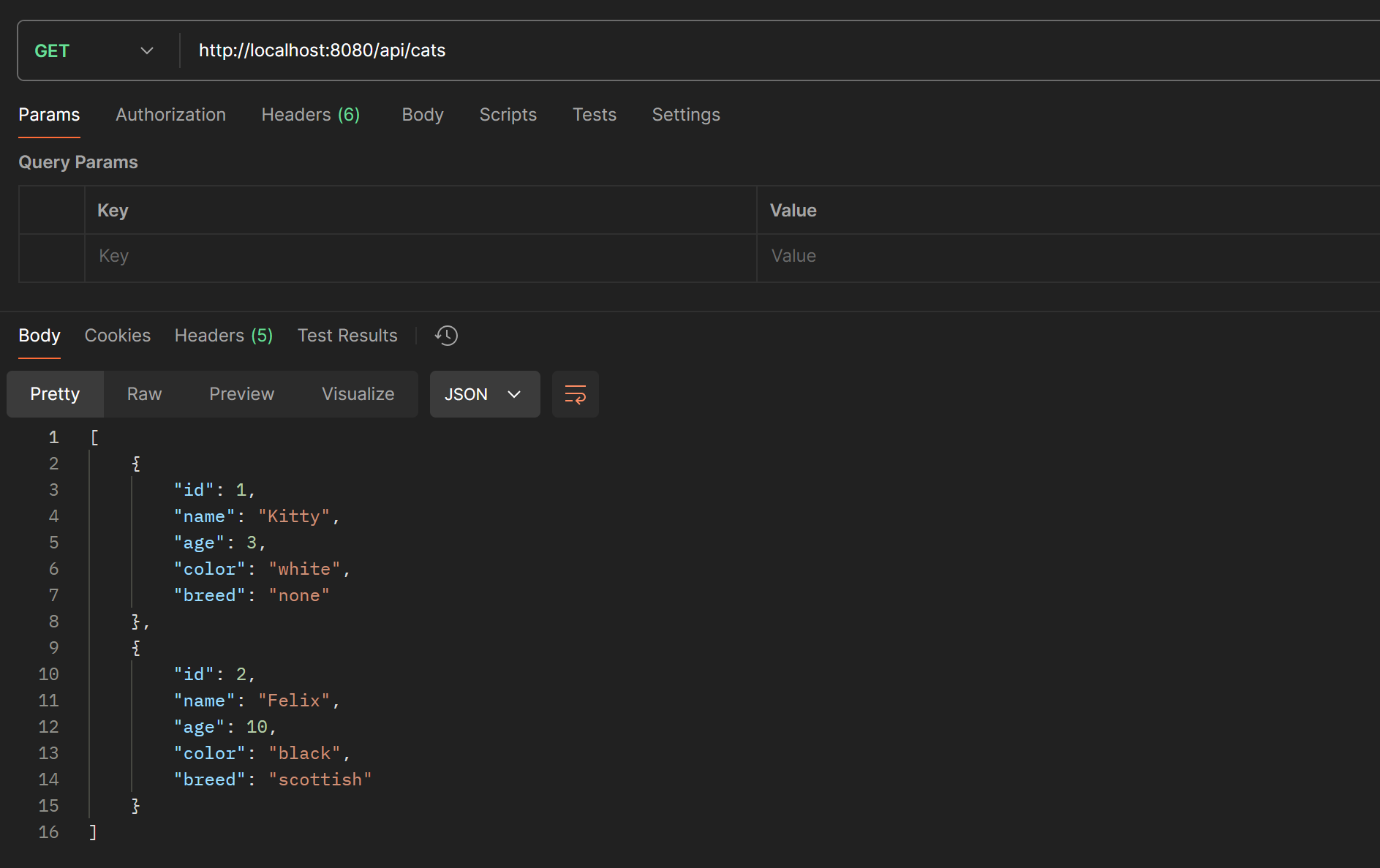


Рисунок 4.2 – Get-запрос

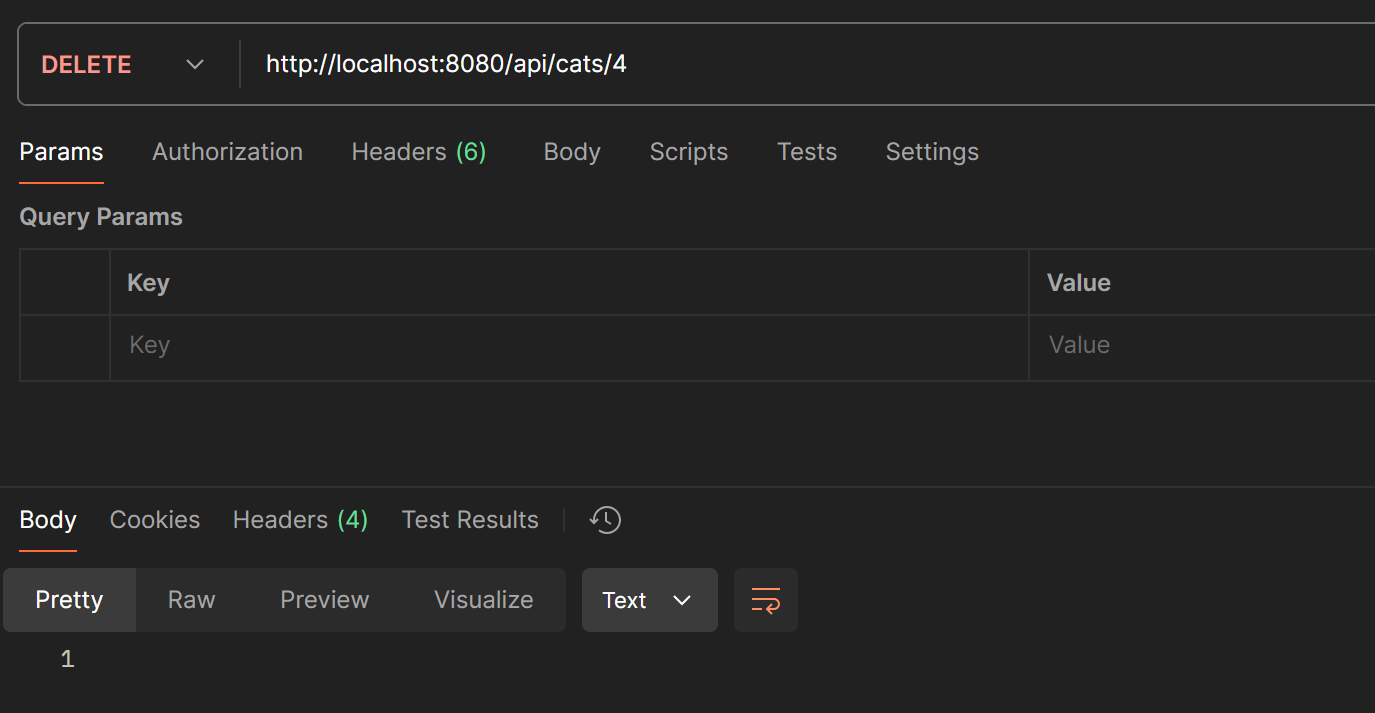


Рисунок 4.3 – Delete-запрос

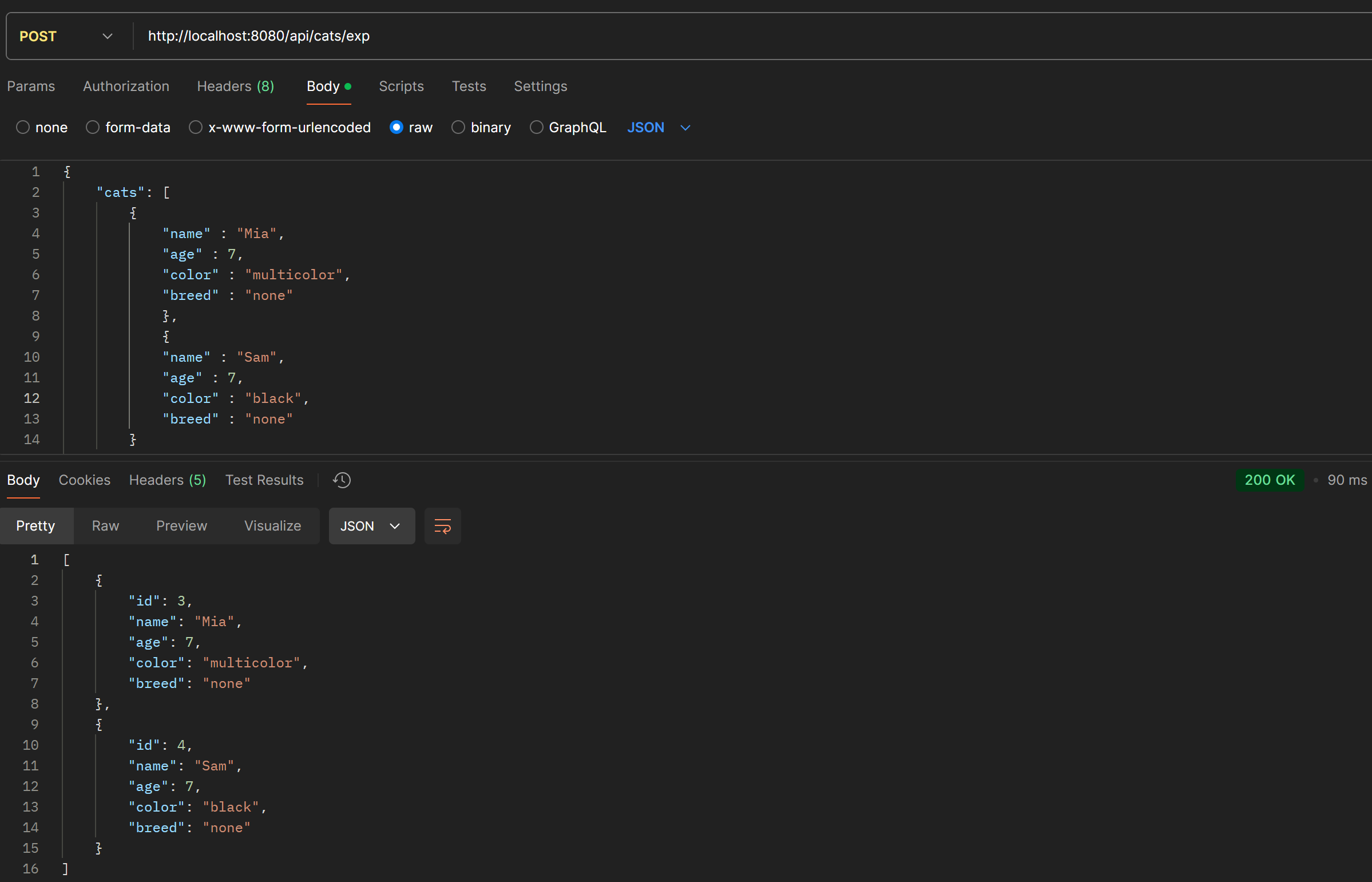


Рисунок 4.4 – Post-запрос на массив данных

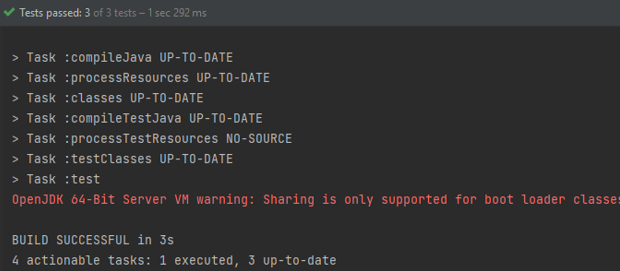


Рисунок 4.6 – Юнит тестирование

**Вывод**

В ходе работы спроектировали, разработали и протестировали клиент-серверную систему с использованием протокола RSocket.

Код на github, подтверждение выполнения работы

# ПРАКТИЧЕСКАЯ РАБОТА №5

**Цель практической работы**: развить у студентов навыки проектирования, разработки и развертывания децентрализованных распределенных систем с использованием современных технологий Spring/Spring Boot и Docker. Студенты научатся обеспечивать надежность и отказоустойчивость систем, а также применять принципы контейнеризации для развертывания многосервисных приложений.

**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.

**Ход работы**

Листинг 5.1 – Модель файла

package com.example.pr5;  
  
import lombok.Data;  
  
@Data  
public class FileInfo {  
 private String filename;  
 private String url;  
}

Листинг 5.2 – Контроллер

package com.example.pr5;  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.web.multipart.MultipartFile;  
import org.springframework.core.io.FileSystemResource;  
import org.springframework.http.ResponseEntity;  
import org.springframework.http.HttpHeaders;  
  
import java.io.File;  
import java.io.IOException;  
  
@Controller  
@RequestMapping("/files")  
@RequiredArgsConstructor  
public class FileController {  
  
 private final FileService fileService;  
  
 @GetMapping("/")  
 public String showUploadForm(Model model) {  
 model.addAttribute("files", fileService.getFiles());  
 return "index";  
 }  
  
 @PostMapping("/upload")  
 public String uploadFile(@RequestParam("file") MultipartFile file, Model model) {  
 try {  
 fileService.saveFile(file);  
 model.addAttribute("uploadStatus", "File uploaded successfully: " + file.getOriginalFilename());  
  
 } catch (IOException e) {  
 model.addAttribute("uploadStatus", "File upload failed: " + e.getMessage());  
 }  
 model.addAttribute("files", fileService.getFiles());  
 return "index";  
 }  
  
 @GetMapping("/download/{filename}")  
 @ResponseBody  
 public ResponseEntity<FileSystemResource> downloadFile(@PathVariable String filename) {  
 File file = fileService.getFile(filename);  
 FileSystemResource resource = new FileSystemResource(file);  
 return ResponseEntity.*ok*()  
 .header(HttpHeaders.*CONTENT\_DISPOSITION*, "attachment; filename=\"" + file.getName() + "\"")  
 .body(resource);  
 }  
}

Листинг 5.3 – Сервис

package com.example.pr5;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Service;  
import org.springframework.web.multipart.MultipartFile;  
  
import java.io.File;  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.nio.file.Paths;  
import java.util.ArrayList;  
import java.util.List;  
  
@Service  
public class FileService {  
// private final String fileStorageLocation = System.getProperty("user.dir") + "/src/main/resources/static/images/";  
 private final String fileStorageLocation = "/upload-files/";  
  
 public FileService() {  
 // Создаем директорию для хранения загруженных файлов  
 new File(fileStorageLocation).mkdirs();  
 }  
  
 public void saveFile(MultipartFile file) throws IOException {  
 Path path = Paths.*get*(fileStorageLocation + file.getOriginalFilename());  
 Files.*copy*(file.getInputStream(), path);  
 }  
  
 public File getFile(String filename) {  
 return new File(fileStorageLocation + filename);  
 }  
  
 public List<String> getFiles() {  
 List<String> files = new ArrayList<>();  
 File folder = new File(fileStorageLocation);  
 for (final File fileEntry : folder.listFiles()) {  
 files.add(fileEntry.getName());  
 }  
 return files;  
 }  
}

Листинг 5.4 – Файл представления

<!DOCTYPE html>  
<html lang="en" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>File Upload & Download</title>  
 <style>  
 body {  
 font-family: Arial, sans-serif;  
 padding: 20px;  
 max-width: 600px;  
 margin: auto;  
 }  
  
 h1, h2 {  
 color: #333;  
 }  
  
 input[type="file"], input[type="text"] {  
 width: 100%;  
 margin-bottom: 10px;  
 }  
 </style>  
</head>  
<body>  
<h1>File Upload & Download</h1>  
  
<h2>Upload File</h2>  
<form th:action="@{/files/upload}" method="post" enctype="multipart/form-data">  
 <input type="file" name="file" required>  
 <button type="submit">загрузить</button>  
</form>  
<p th:text="${uploadStatus}"></p>  
<h2>Список загруженных файлов</h2>  
<ul>  
 <th:block th:each="file : ${files}">  
 <li>  
 <a th:href="@{|/files/download/${file}|}" th:text="${file}" ></a>  
 </li>  
 </th:block>  
</ul>  
  
</body>  
</html>

Листинг 5.5 - Dockerfile java-приложения

FROM maven:latest  
  
ADD . /app  
WORKDIR /app  
  
RUN mvn clean install -DskipTests  
  
FROM openjdk:17-oracle  
  
LABEL name="ZEA"  
  
ARG *JAR\_FILE*=/app/target/\*.jar  
COPY --from=*0* ${*JAR\_FILE*} /application.jar  
ENTRYPOINT ["java", "-jar", "application.jar"]

Листинг 5.6 - Конфигурация nginx (nginx.conf)

upstream myapp {  
 server app1:8080;  
}  
  
server {  
 listen 80;  
  
 location ^~ / {  
 proxy\_pass http://myapp;  
 }  
}

Листинг 5.7 - docker-compose.yaml

version: "3.9"  
services:  
 app1:  
 build: .  
 restart:  
 on-failure  
 environment:  
 - NAME\_APP=app1  
 volumes:  
 - ./upload-files:/upload-files  
 deploy:  
 mode: replicated  
 replicas: 4  
  
 proxy:  
 image: nginx:latest  
 ports:  
 - '80:80'  
 volumes:  
 - ./nginx.conf:/etc/nginx/conf.d/default.conf

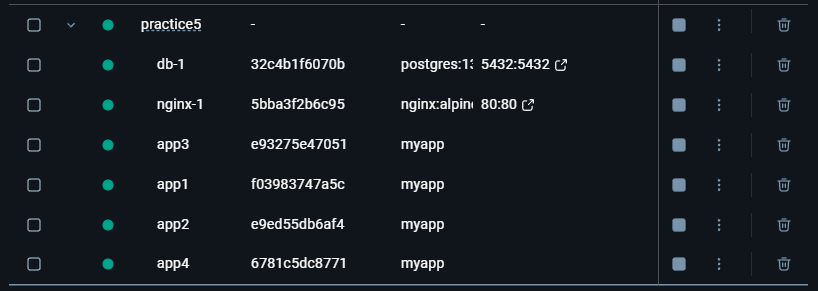


Рисунок 5.1 – Докер контейнеры

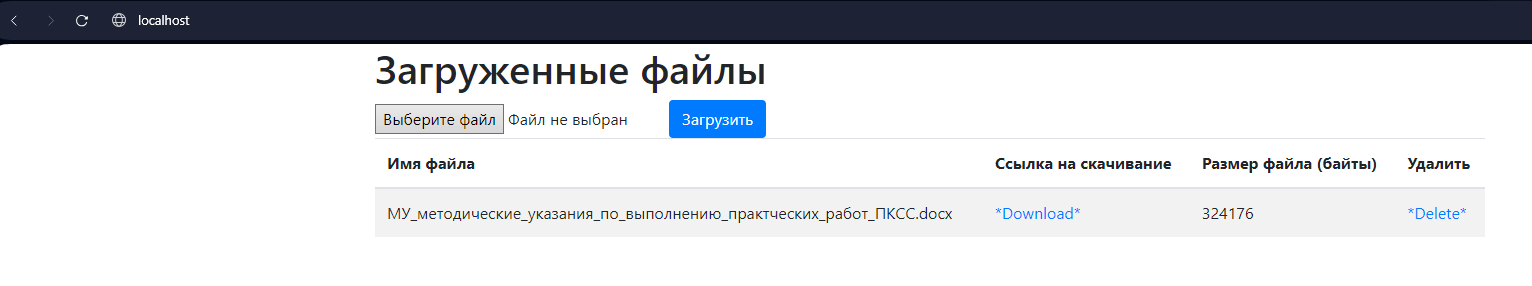


Рисунок 5.2 – Работающее приложение

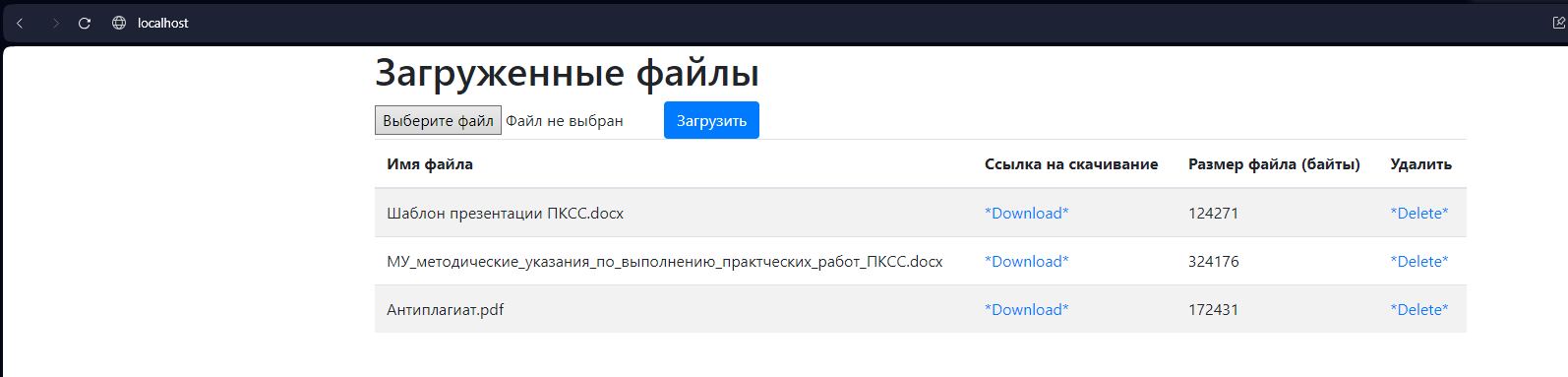


Рисунок 5.3 – Проверка работоспособности при выключении 2ух контейнеров

**Вывод**

В ходе выполнения работы было спроектировано, разработано и развернуто децентрализованная распределенная система с использованием современных технологий Spring/Spring Boot и Docker. Было развернуто 4 реплики приложения.

Код на github, подтверждение выполнения работы

# ПРАКТИЧЕСКАЯ РАБОТА №6

**Цель практической работы:** развить у студентов навыки разработки и деплоя смарт-контрактов, а также тестирования и демонстрации их работы.

**Задание**

Написать и задеплоить смарт-контракты моделирующие процессы покупки и обмена ценными бумагами (возможно выполнение индивидуального варианта).

Написать unit-тесты.

Продемонстрировать работу через встроенные инструменты Remix IDE.

**Ход работы**

Листинг 6.1 – Смарт-контракт

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract SimpleStore {

address public owner;

mapping(uint256 => Product) public products;

uint256 public productCount = 0;

struct Product {

string name;

uint256 price;

uint256 quantity;

}

event ProductAdded(uint256 productId, string name, uint256 price, uint256 quantity);

event ProductPurchased(uint256 productId, uint256 quantity, uint256 totalCost);

modifier onlyOwner() {

require(msg.sender == owner, "Only the owner can call this function");

\_;

}

constructor() {

owner = msg.sender;

}

// Добавление продукта владельцем

function addProduct(string memory \_name, uint256 \_price, uint256 \_quantity) public onlyOwner {

productCount++;

products[productCount] = Product(\_name, \_price, \_quantity);

emit ProductAdded(productCount, \_name, \_price, \_quantity);

}

// Покупка продукта

Продолжение Листинга 6.1

function purchaseProduct(uint256 \_productId, uint256 \_quantity) public payable {

require(\_productId > 0 && \_productId <= productCount, "Invalid product ID");

Product storage product = products[\_productId];

require(product.quantity >= \_quantity, "Not enough stock available");

uint256 totalCost = product.price \* \_quantity;

require(msg.value >= totalCost, "Insufficient funds sent");

// Обновляем количество продукта

product.quantity -= \_quantity;

// Переводим излишек обратно покупателю, если был отправлен излишек

if (msg.value > totalCost) {

payable(msg.sender).transfer(msg.value - totalCost);

}

emit ProductPurchased(\_productId, \_quantity, totalCost);

}

// Вывод средств владельцем контракта

function withdrawBalance() public onlyOwner {

uint256 contractBalance = address(this).balance;

require(contractBalance > 0, "No balance to withdraw");

payable(owner).transfer(contractBalance);

}

}

Листинг 6.2 - Тесты

const { expect } = require("chai");

const { ethers } = require("hardhat");

describe("SimpleStore Contract", function () {

let simpleStore;

let owner;

let addr1;

let addr2;

let productId;

beforeEach(async function () {

[owner, addr1, addr2] = await ethers.getSigners();

const SimpleStore = await ethers.getContractFactory("SimpleStore");

simpleStore = await SimpleStore.deploy();

await simpleStore.deployed();

});

// Тест на добавление продукта

it("should add a product successfully", async function () {

await simpleStore.connect(owner).addProduct("Product A", ethers.utils.parseEther("0.1"), 10); // Цена 0.1 ETH

const product = await simpleStore.products(1);

expect(product.name).to.equal("Product A");

expect(product.price).to.equal(ethers.utils.parseEther("0.1"));

expect(product.quantity).to.equal(10);

await expect(

simpleStore.connect(owner).addProduct("Product B", ethers.utils.parseEther("0.15"), 5)

)

.to.emit(simpleStore, "ProductAdded")

Продолжение листинга 6.2

|  |
| --- |
| .withArgs(2, "Product B", ethers.utils.parseEther("0.15"), 5);  });  // Тест на покупку продукта  it("should allow a user to purchase a product", async function () {  await simpleStore.connect(owner).addProduct("Product A", ethers.utils.parseEther("0.1"), 10); // 0.1 ETH цена  productId = 1;  await simpleStore.connect(addr1).purchaseProduct(productId, 2, {  value: ethers.utils.parseEther("0.2"), // Отправляем 0.2 ETH (0.1 \* 2 = 0.2 ETH)  });  const product = await simpleStore.products(productId);  expect(product.quantity).to.equal(8); // 10 - 2  await expect(  simpleStore.connect(addr1).purchaseProduct(productId, 1, {  value: ethers.utils.parseEther("0.1"), // 0.1 \* 1 = 0.1 ETH  }) |

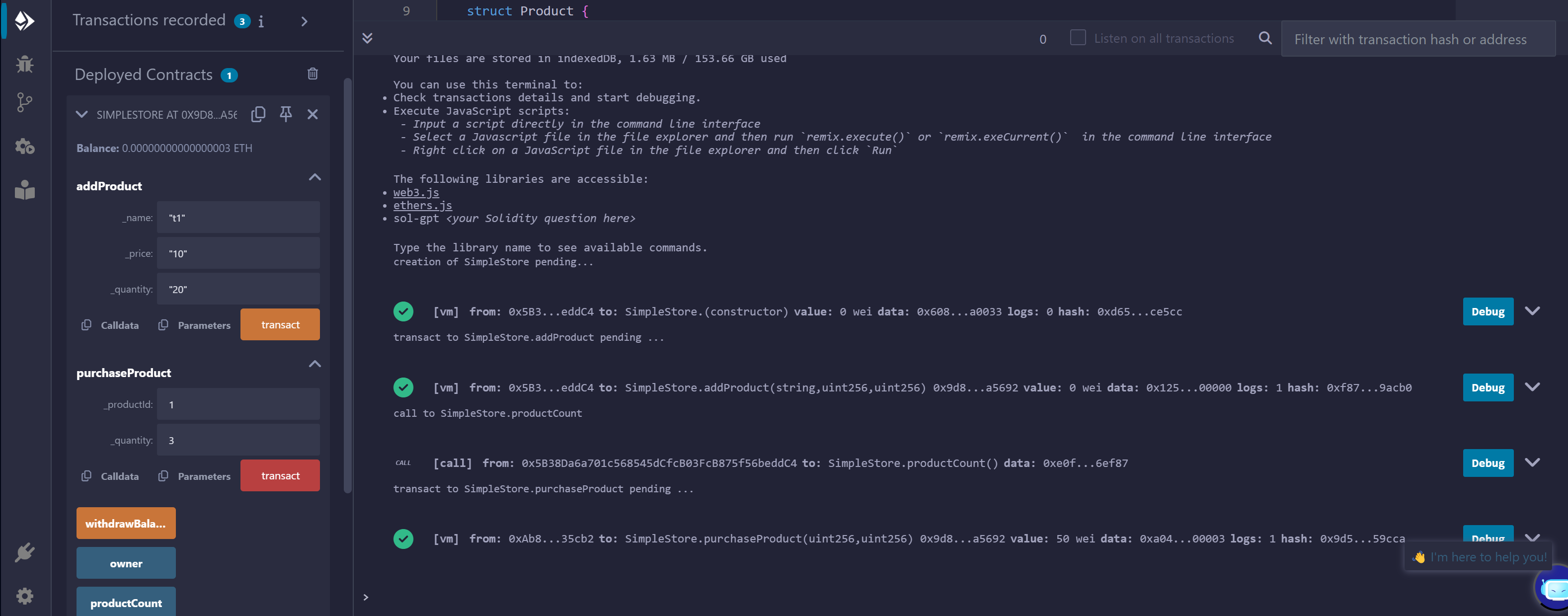
****

Рисунок 6.1 – Пройденные тесты

**Вывод**

В ходе выполнения работы были написаны и задеплоены смарт-контракты моделирующие процессы покупки и обмена ценными бумагами.

Код на github, подтверждение выполнения работы

# ПРАКТИЧЕСКАЯ РАБОТА №7

**Цель практической работы:** развить у студентов навыки разработки реактивных REST API с использованием Spring WebFlux. В ходе выполнения задания студенты освоят создание и тестирование эндпоинтов, работающих с реактивными типами данных, научатся применять операторы преобразования потоков, работать с базами данных в асинхронном режиме, а также обеспечивать управление потоком данных (backpressure) и обработку ошибок.

**Задание**

Реализовать сервис, предоставляющий реактивный REST API с использованием 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.

**Ход работы**

Листинг 7.1 – application.properties

spring.application.name=pr7

spring.r2dbc.url=r2dbc:postgresql://localhost:54321/pr7

spring.r2dbc.username=postgres

spring.r2dbc.password=password

logging.level.org.springframework.r2dbc=DEBUG

server.port=8081

Листинг 7.2 – Модель

@NoArgsConstructor

@AllArgsConstructor

@Data

@Table("cats")

public class Cat {

@Id

private Long id;

@Column("name")

private String name;

@Column("color")

private String color;

@Column("age")

private Integer age;

@Column("breed")

private String breed;

}

Листинг 7.3 – init.sql скрипт

CREATE TABLE IF NOT EXISTS cats (

id SERIAL PRIMARY KEY,

name VARCHAR(255),

color VARCHAR(255),

age INTEGER,

breed VARCHAR(255)

);

Листинг 7.4 – Конфигурация запуска скрипта

@Configuration

public class DbSchemaInitOnStartup {

@Bean

ConnectionFactoryInitializer initializer(@Qualifier("connectionFactory")

ConnectionFactory connectionFactory) {

ConnectionFactoryInitializer initializer = new ConnectionFactoryInitializer();

initializer.setConnectionFactory(connectionFactory);

ResourceDatabasePopulator resource = new ResourceDatabasePopulator(new

ClassPathResource("init.sql"));

initializer.setDatabasePopulator(resource);

return initializer; }

}

Листинг 7.5 – Reactive репозиторий модели

@Repository

public interface CatRepository extends R2dbcRepository<Cat, Long> {

}

Листинг 7.6 – WebFlux контроллер

@RestController

@RequestMapping("/cats")

public class CatController {

private final CatRepository catRepository;

@Autowired

public CatController(CatRepository catRepository) {

this.catRepository = catRepository;

}

Продолжение листинга 7.6

@GetMapping("/{id}")

public Mono<ResponseEntity<Cat>> getCatById(@PathVariable Long id) {

return catRepository.findById(id)

.map(cat -> ResponseEntity.ok(cat))

.defaultIfEmpty(ResponseEntity.notFound().build());

}

@GetMapping

public Flux<Cat> getAllCats(@RequestParam(name = "minage", required = false) Integer minAge) {

Flux<Cat> cats = catRepository.findAll();

if (minAge != null && minAge > 0)

{

cats = cats.filter(cat -> cat.getAge() >= minAge);

}

return cats

.map(this::transformCat)

.onErrorResume(e -> {

return Flux.error(new CustomException("Failed to fetch cats", e));

})

.onBackpressureBuffer(); // Работа в формате backpressure

}

@PostMapping

@ResponseStatus(HttpStatus.CREATED)

public Mono<Cat> createCat(@RequestBody Cat cat) {

return catRepository.save(cat);

}

@PutMapping("/{id}")

public Mono<ResponseEntity<Cat>> updateCat(@PathVariable Long id, @RequestBody Cat updatedCat) {

return catRepository.findById(id)

.flatMap(existingCat -> {

existingCat.setName(updatedCat.getName());

existingCat.setColor(updatedCat.getColor());

existingCat.setAge(updatedCat.getAge());

existingCat.setBreed(updatedCat.getBreed());

return catRepository.save(existingCat);

})

.map(ResponseEntity::ok)

.defaultIfEmpty(ResponseEntity.notFound().build());

}

@DeleteMapping("/{id}")

public Mono<ResponseEntity<Void>> deleteCat(@PathVariable Long id) {

return catRepository.findById(id)

.flatMap(existingCat -> catRepository

.delete(existingCat).then(Mono.just(new ResponseEntity<Void>(HttpStatus.NO\_CONTENT)))

)

.defaultIfEmpty(ResponseEntity.notFound().build());

}

private Cat transformCat(Cat cat) {

cat.setName(cat.getName().toUpperCase());

return cat;

}

}

Листинг 7.7 – Unit-тесты

public class CatControllerTest {

@Test

public void testCreateCat() {

Cat cat = new Cat();

cat.setId(1L);

cat.setName("Asteroid destroyer");

CatRepository catRepository = Mockito.mock(CatRepository.class);

when(catRepository.save(cat)).thenReturn(Mono.just(cat));

CatController catController = new CatController(catRepository);

Mono<Cat> response = catController.createCat(cat);

assertEquals(cat, response.block());

}

@Test

public void testGetAllCats() {

Cat cat1 = new Cat();

cat1.setId(1L);

cat1.setName("Asteroid destroyer");

cat1.setAge(3);

Cat cat2 = new Cat();

cat2.setId(2L);

cat2.setName("Batman`s granny");

cat2.setAge(4);

CatRepository catRepository = Mockito.mock(CatRepository.class);

when(catRepository.findAll()).thenReturn(Flux.just(cat1, cat2));

CatController catController = new CatController(catRepository);

Flux<Cat> response = catController.getAllCats(null);

assertEquals(2, response.collectList().block().size());

}

@Test

public void testDeleteCat() {

Cat cat = new Cat();

cat.setId(1L);

cat.setName("Asteroid destroyer");

CatRepository catRepository = Mockito.mock(CatRepository.class);

when(catRepository.findById(1L)).thenReturn(Mono.just(cat));

when(catRepository.delete(cat)).thenReturn(Mono.empty());

CatController catController = new CatController(catRepository);

ResponseEntity<Void> response = catController.deleteCat(1L).block();

assertEquals(HttpStatus.NO\_CONTENT, response.getStatusCode());

}

}

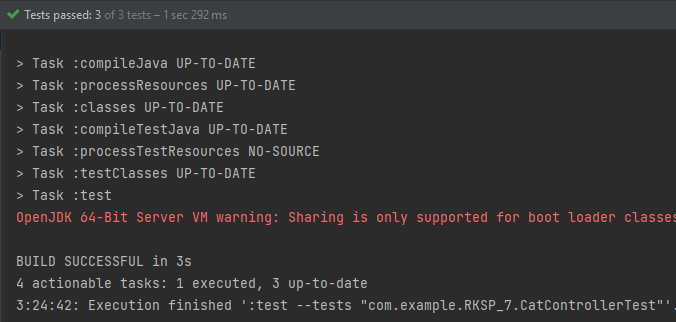


Рисунок 7.1 – Пройденные тесты

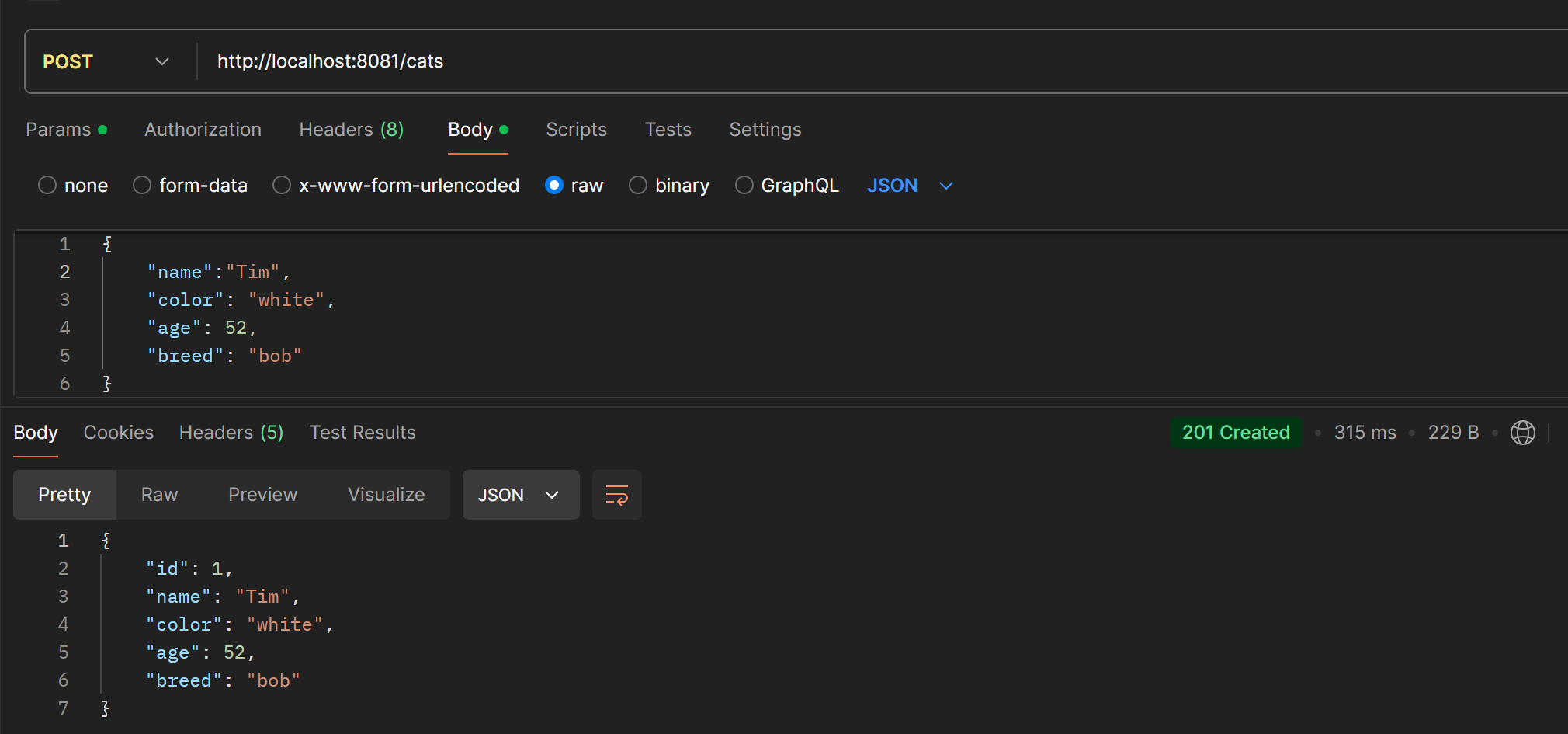


Рисунок 7.2 – Добавление кота

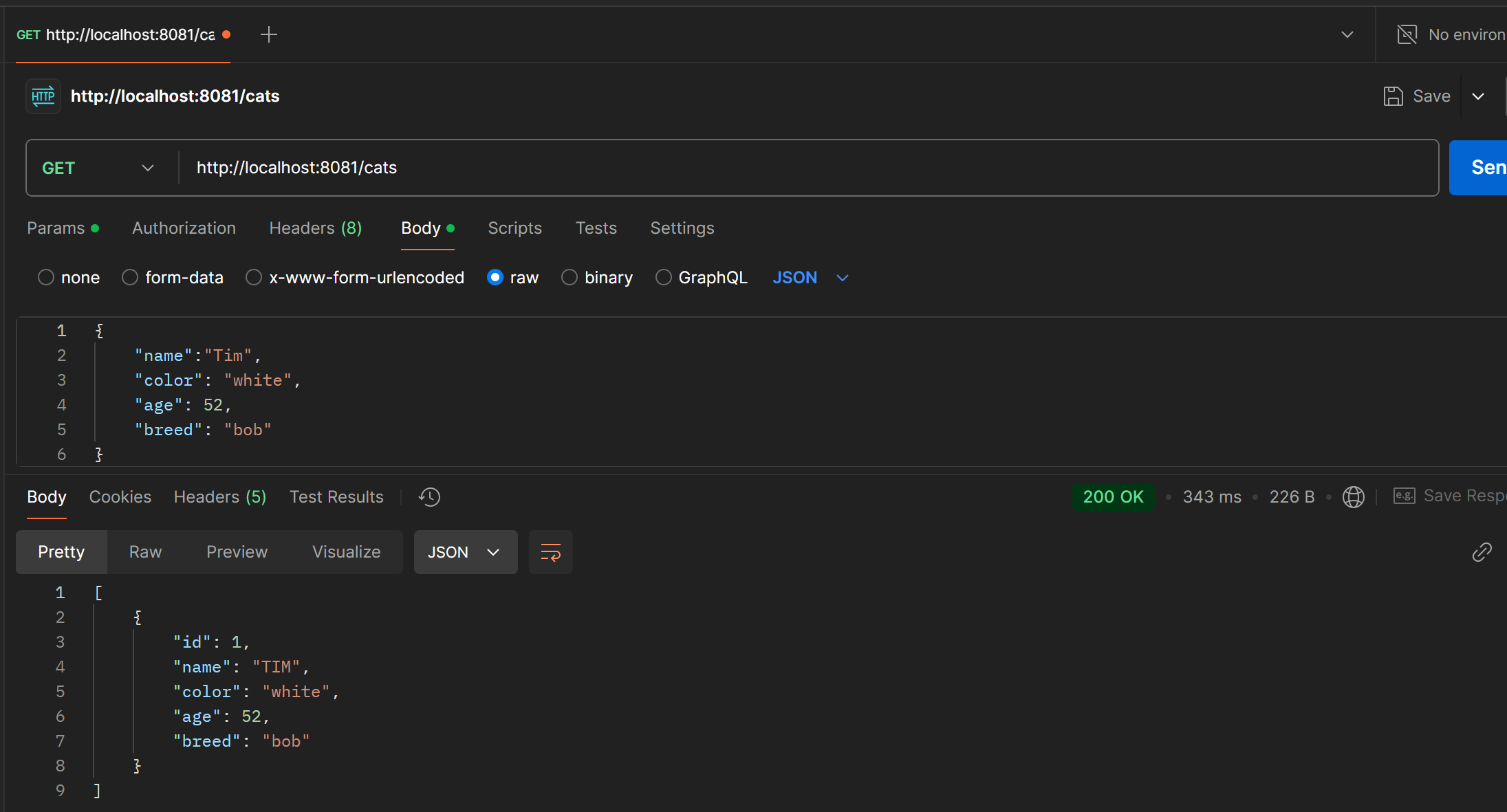


Рисунок 7.3 – получение списка котов

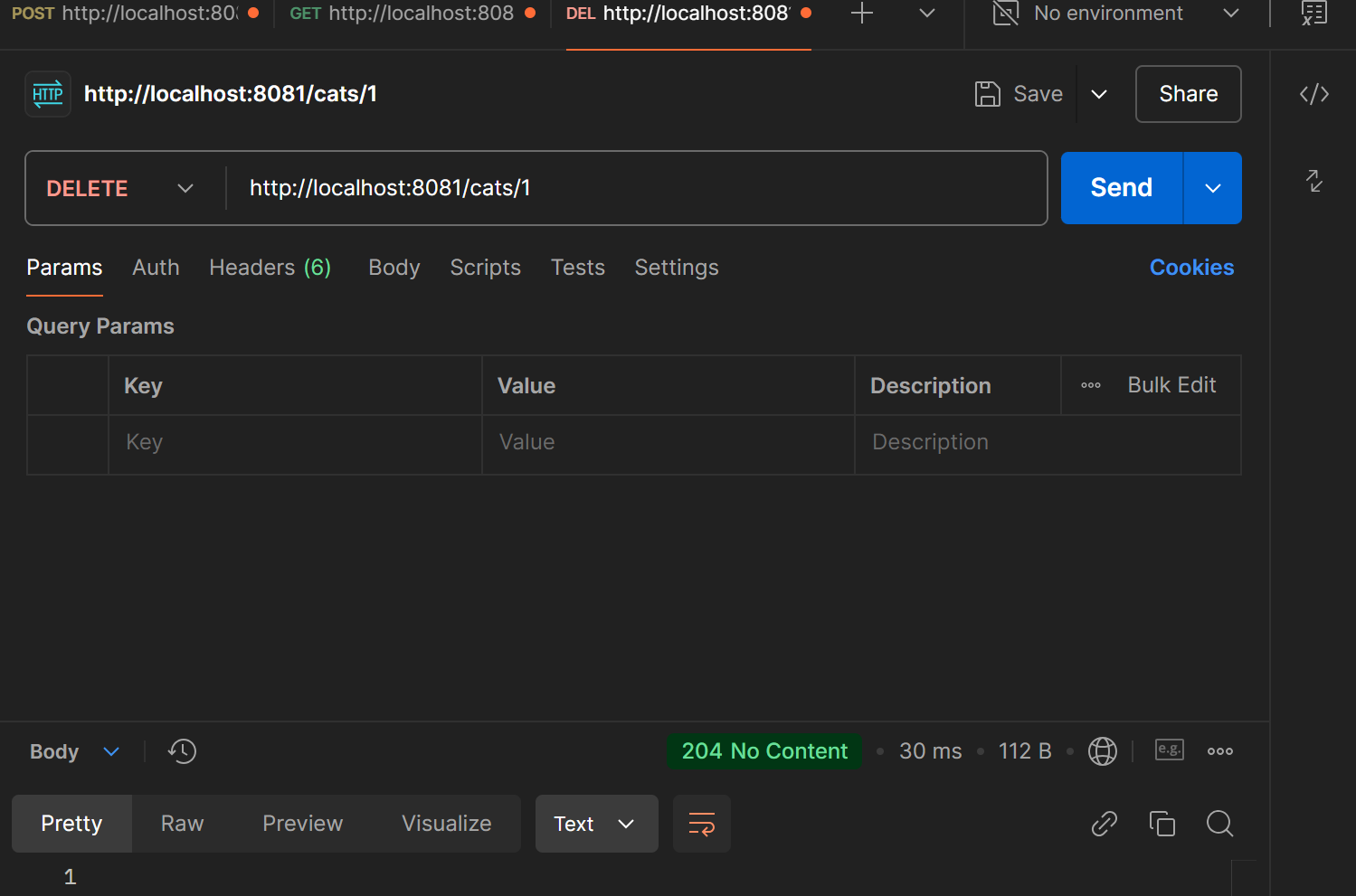


Рисунок 7.4 – удаление кота

**Выводы**

В ходе работыбыл реализован сервис, предоставляющий реактивный REST API с использованием Spring WebFlux.

Рисунок 7.8 – Код на github

# ПРАКТИЧЕСКАЯ РАБОТА №8

**Цель практической 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**

Разработать микросервисную систему с использованием инструментов Spring Cloud (возможно использование аналогов). Предметная область разрабатываемой системы выбирается студентом самостоятельно, но должна быть уникальна в рамках учебной группы. Система должна удовлетворять следующим требованиям:

⎯ Реализована авторизация при помощи протокола OAuth2;

⎯ Реализован сервис конфигурации с использованием Git в качестве источника данных (Spring Cloud Config);

⎯ Реализован паттерн API Gateway;

⎯ Реализован паттерн Service Discovery;

⎯ Реализован балансировщик нагрузки;

⎯ Реализовано как минимум 3 микросервиса, связанных с предметной областью;

⎯ Продемонстрировано взаимодействие между микросервисами с использованием FeignClient;

⎯ Микросервисы взаимодействуют с базой данных;

⎯ Для каждого микросервиса используется отдельный экземпляр базы данных;

⎯ Система покрыта Unit-тестами;

⎯ Написаны файлы для развертывания системы (возможно использование kubernetes);

⎯ Система развернута в Yandex(VK) Cloud. В случае невозможности развертывания в облаке допускается использование локального minicube.

**Ход работы**

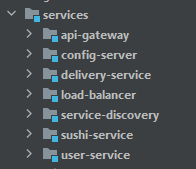
****

Рисунок 8.1 – Список микросервисов

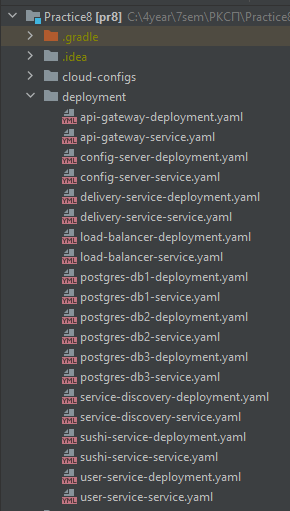


Рисунок 8.2 – список файлов для k8s

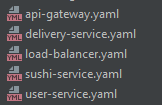


Рисунок 8.3 – Конфиги для spring cloud config server’а

Листинг 8.1 – Пример контроллера сервиса

package org.example;  
  
import jakarta.ws.rs.NotFoundException;  
import lombok.RequiredArgsConstructor;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.Optional;  
  
@RestController  
@RequestMapping("/api/delivery")  
@RequiredArgsConstructor  
public class DeliveryController {  
  
 private final DeliveryRepository deliveryRepository;  
 private final UserClient userClient;  
 private final SushiClient SushiClient;  
  
 @GetMapping("/{id}")  
 public ResponseEntity<DeliveryDTO> getDelivery(@PathVariable("id") Long id) {  
 Optional<Delivery> delivery = deliveryRepository.findById(id);  
 if (delivery.isPresent()) {  
 Sushi Sushi = SushiClient.getSushiById(delivery.get().getSushiId());  
 User user = userClient.getUserById(delivery.get().getUserId());  
 DeliveryDTO deliveryDTO = DeliveryDTO.*builder*()  
 .id(delivery.get().getId())  
 .Sushi(Sushi)  
 .user(user)  
 .build();  
 return ResponseEntity.*ok*(deliveryDTO);  
 } else {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 @PostMapping("/create")  
 public ResponseEntity<Delivery> createDelivery(@RequestParam("SushiId") Long SushiId,  
 @RequestParam("userId") Long userId) {  
 try {  
 Sushi Sushi = SushiClient.getSushiById(SushiId);  
 User user = userClient.getUserById(userId);  
 } catch (NotFoundException ex) {  
 return ResponseEntity.*badRequest*().build();  
 }  
  
 Delivery newDelivery = Delivery.*builder*()  
 .userId(userId)  
 .SushiId(SushiId)  
 .build();  
 return ResponseEntity.*status*(HttpStatus.*CREATED*).body(deliveryRepository.save(newDelivery));  
 }  
}

Листинг 8.2 – Пример репозитория сервиса

package org.example;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface DeliveryRepository extends JpaRepository<Delivery, Long> {  
  
}

Листинг 8.3 – Пример модели сервиса

package org.example;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.Id;  
import jakarta.persistence.Table;  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Entity  
@Table(name="delivery")  
@Builder  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class Delivery {  
 @Id  
 @GeneratedValue  
 private Long id;  
  
 private Long userId;  
  
 private Long SushiId;  
}

Листинг 8.4 - Конфигурация авторизации

spring.security.oauth2.client:  
 registration.github:  
 client-id: \*\*\*\*

ient-secret: \*\*\*\*  
 authorization-grant-type: authorization\_code  
 redirect-uri: "{baseUrl}/login/oauth2/code/{registrationId}"  
 scope: read:user,user:email  
 provider.github:  
 authorization-uri: https://github.com/login/oauth/authorize  
 token-uri: https://github.com/login/oauth/access\_token  
 user-info-uri: https://api.github.com/user  
 user-name-attribute: login

Листинг 8.5 - Конфигурация Spring Cloud Config

server.port: 8888  
spring:  
 application:  
 name: config-server  
 cloud:  
 config:  
 enabled: false  
 config.server.git:  
 search-paths:  
 - cloud-configs  
 clone-on-start: true  
 default-label: main  
 uri: https://github.com/BIGmindede/RKSP-8.git  
 ignore-local-ssh-settings: true  
  
eureka:  
 client:  
 service-url:  
 defaultZone: http://service-discovery:8761/eureka/

Листинг 8.6 - application.properties для Eureka сервера

spring.application.name: service-discovery  
server.port: 8761  
  
eureka.client:  
 register-with-eureka: false  
 fetch-registry: false  
logging:  
 level:  
 com.netflix.eureka: OFF  
 com.netflix.discovery: OFF

Листинг 8.7 - Настройка балансировки

server.port: 9000  
  
spring.cloud.loadbalancer.clients:  
 delivery-service:  
 instances:  
 - uri: http://delivery-service:8082  
  
eureka.client.service-url.defaultZone: http://service-discovery:8761/eureka/  
  
delivery.service.url: http://delivery-service:8082

Листинг 8.8 – Пример использования FeignClient

package org.example;  
  
import org.springframework.cloud.openfeign.FeignClient;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
  
@FeignClient(name = "SushiClient", url = "${Sushi.service.url}")  
public interface SushiClient {  
 @GetMapping("/api/Sushi/{id}")  
 Sushi getSushiById(@PathVariable Long id);  
}

Листинг 8.9 – Пример docker образа

FROM amazoncorretto:21.0.5-alpine  
WORKDIR /app  
COPY /build/libs/api-gateway.jar /app  
CMD ["java", "-jar", "api-gateway.jar"]  
EXPOSE 80

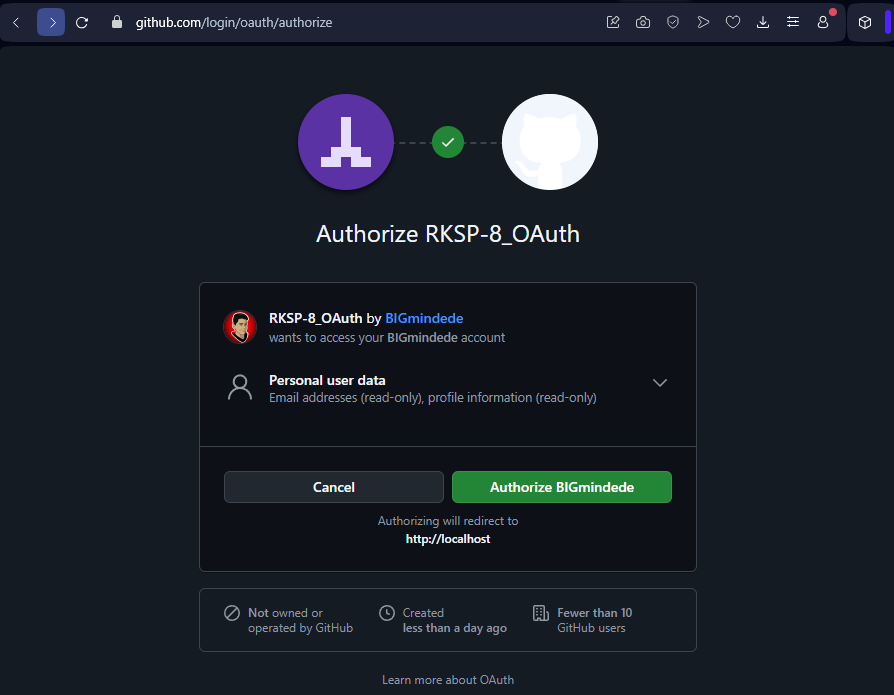


Рисунок 8.4 – oauth 2.0 авторизация через github

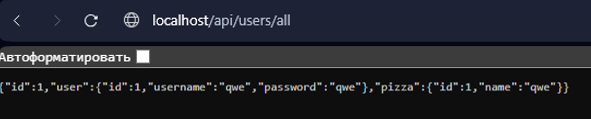


Рисунок 8.5 – Выполнение запроса после авторизации

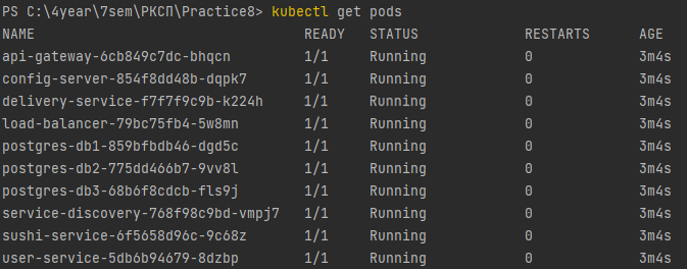


Рисунок 8.6 – Развернутые поды

Листинг 8.10 – код тестов

import org.example.\*;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.junit.jupiter.MockitoExtension;  
import org.springframework.http.ResponseEntity;  
import java.util.Optional;  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.Mockito.\*;  
  
@ExtendWith(MockitoExtension.class)  
public class DeliveryControllerTest {  
  
 @Mock  
 private DeliveryRepository deliveryRepository;  
  
 @Mock  
 private UserClient userClient;  
  
 @Mock  
 private SushiClient SushiClient;  
  
 @InjectMocks  
 private DeliveryController deliveryController;  
  
 private Delivery delivery;  
 private Sushi Sushi;  
 private User user;  
  
 @BeforeEach  
 void setUp() {  
 delivery = Delivery.*builder*()  
 .id(1L)  
 .userId(10L)  
 .SushiId(20L)  
 .build();  
  
 Sushi = Sushi.*builder*()  
 .id(20L)  
 .name("Margherita")  
 .build();  
  
 user = User.*builder*()  
 .id(10L)  
 .username("john\_doe")  
 .password("password123")  
 .build();  
 }  
  
 @Test  
 void getDelivery\_shouldReturnDeliveryDTO\_whenDeliveryExists() {  
 *when*(deliveryRepository.findById(1L)).thenReturn(Optional.*of*(delivery));  
 *when*(SushiClient.getSushiById(20L)).thenReturn(Sushi);  
 *when*(userClient.getUserById(10L)).thenReturn(user);  
  
 ResponseEntity<DeliveryDTO> response = deliveryController.getDelivery(1L);  
  
 *assertEquals*(200, response.getStatusCodeValue());  
 *assertNotNull*(response.getBody());  
 *assertEquals*(1L, response.getBody().getId());  
 *assertEquals*("Margherita", response.getBody().getSushi().getName());  
 *assertEquals*("john\_doe", response.getBody().getUser().getUsername());  
  
 *verify*(deliveryRepository, *times*(1)).findById(1L);  
 *verify*(SushiClient, *times*(1)).getSushiById(20L);  
 *verify*(userClient, *times*(1)).getUserById(10L);  
 }  
  
 @Test  
 void getDelivery\_shouldReturnNotFound\_whenDeliveryDoesNotExist() {  
 *when*(deliveryRepository.findById(1L)).thenReturn(Optional.*empty*());  
  
 ResponseEntity<DeliveryDTO> response = deliveryController.getDelivery(1L);  
  
 *assertEquals*(404, response.getStatusCodeValue());  
 *assertNull*(response.getBody());  
  
 *verify*(deliveryRepository, *times*(1)).findById(1L);  
 *verifyNoInteractions*(SushiClient, userClient);  
 }  
  
 @Test  
 void createDelivery\_shouldReturnCreatedDelivery\_whenValidData() {  
 *when*(SushiClient.getSushiById(20L)).thenReturn(Sushi);  
 *when*(userClient.getUserById(10L)).thenReturn(user);  
 *when*(deliveryRepository.save(*any*(Delivery.class))).thenReturn(delivery);  
  
 ResponseEntity<Delivery> response = deliveryController.createDelivery(20L, 10L);  
  
 *assertEquals*(201, response.getStatusCodeValue());  
 *assertNotNull*(response.getBody());  
 *assertEquals*(1L, response.getBody().getId());  
 *assertEquals*(10L, response.getBody().getUserId());  
 *assertEquals*(20L, response.getBody().getSushiId());  
  
 *verify*(SushiClient, *times*(1)).getSushiById(20L);  
 *verify*(userClient, *times*(1)).getUserById(10L);  
 *verify*(deliveryRepository, *times*(1)).save(*any*(Delivery.class));  
 }  
}

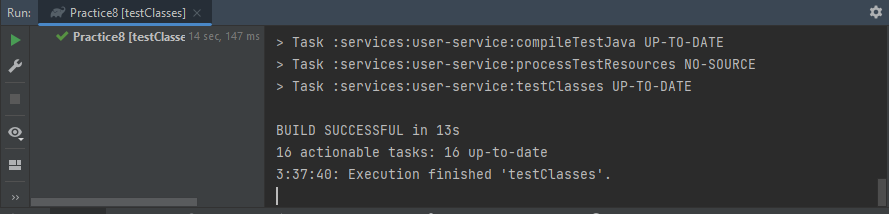


Рисунок 8.7 – Пройденные тесты

**Вывод**

В ходе работы была разработана микросервисная система с использованием инструментов Spring Cloud.

Код на github, подтверждение выполнения работы

# СПИСОК **ИСПОЛЬЗОВАННЫХ** ИСТОЧНИКОВ

1. Java NIO Programming Cookbook// https://www.appservgrid.com/codegeeks/Java-NIO-Programming-Cookbook.pdf
2. Reactive Programming with RxJava: Creating Asynchronous, Event-Based Applications. By Tomasz Nurkiewicz,‎ Ben Christensen. O'Reilly Media; 1 edition October 27, 2016; 372 pages; ISBN-13: 978-1491931653
3. Reactive Programming Third Edition Paperback by Gerardus Blokdyk
4. Java Concurrency на практике (2020) Автор: Брайан Гетц
5. Data Science with Java by Michael Brzustowicz
6. Introducing Blockchain with Java Автор: Spiro Buzharovski // <https://coderbooks.ru/books/java/introducing-blockchain-with-java/>
7. Реактивное программирование со Spring, часть 2 Project Reactor // <https://habr.com/ru/articles/565050/>
8. Spring Boot in Action Paperback – Illustrated by Craig Walls
9. Spring Microservices in Action Paperback by John Carnell
10. Cloud Native Java: Designing Resilient Systems with Spring Boot, Spring Cloud, and Cloud Foundry Paperback by Josh Long, Kenny Bastani
11. Документация по Java и многопоточности [<https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html>]
12. Введение в реактивное программирование с использованием RxJava [https://habr.com/ru/post/346782/]
13. RxJava для начинающих [https://javarush.com/groups/posts/2635-rxjava-dlja-nachinajushhikh]
14. Официальная документация Spring [<https://spring.io/guides>]
15. Spring Boot: Пошаговое руководство на русском [https://spring-projects.ru/guides/]
16. Введение в RSocket [https://habr.com/ru/company/oleg-bunin/blog/519844/]
17. RSocket и его применение в микросервисах [https://www.baeldung.com/rsocket]
18. Официальная документация Docker на русском [https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/]
19. Docker: Полное руководство для начинающих [https://habr.com/ru/company/flant/blog/490404/]
20. Пример использования WatchService на Java [https://javarush.com/groups/posts/705-nablyudenie-za-izmeneniyami-v-faylovoy-sisteme]
21. Что такое смарт-контракты и как они работают [https://forklog.com/primer-na-smart-kontrakty/]
22. Введение в разработку смарт-контрактов на Solidity [https://habr.com/ru/post/431396/]
23. Реактивное программирование с использованием Spring WebFlux [https://habr.com/ru/company/otus/blog/544540/]
24. Работа с реактивными типами данных в Spring WebFlux [https://javarush.com/groups/posts/razrabotka-reactivnih-web-api-s-ispolzovaniem-spring-webflux]
25. Руководство по созданию микросервисов с использованием Spring Cloud [https://spring-projects.ru/guides/microservices/]
26. Паттерны микросервисов: API Gateway и Service Discovery [https://habr.com/ru/post/431654/]