|  |
| --- |
| https://lh6.googleusercontent.com/QcftzNtI05T0Y6fjdSh1Rr2rt8oqZ1IvnLvbn1jLJ7CCyteVir3k-xBLv4SL1wAgWJsRhmmJSR0UW-RP63_GQenE4vVWv05BRoZTsmIcBccVTnfxwmsnNMvjg599x9SqZd8E3dkd |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **«МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** |

Институт информационных технологий (ИТ)

Кафедра инструментального и прикладного программного обеспечения (ИиППО)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКИМ РАБОТАМ** | | | | | |
| **по дисциплине «Разработка клиент-серверных приложений»** | | | | | |
|  | | | | | |
|  | | | | | |
| Выполнил студент группы ИКБО-20-19 | | | | Московка А.А. | |
|  | | | |  | |
| Принял  *Ассистент* | | Зарипов Е.А. | | |
| Практические работы выполнены | «\_\_\_»\_\_\_\_\_\_\_2022 г. | | (подпись студента) | | |
| «Зачтено» | «\_\_\_»\_\_\_\_\_\_\_2022 г. | | (подпись руководителя) | | |
|  |  | |  | | |

Москва 2022

Содержание

[Практическая работа №1: Примитивы синхронизации и их использование при создании клиент серверных приложений 3](#_Toc121953345)

[Практическая работа №2: Java NIO 14](#_Toc121953346)

[Практическая работа №3. RxJava 25](#_Toc121953347)

[Практическая работа №4. Rsocket 35](#_Toc121953348)

[Практическая работа №5. Decentralized Distributed System 41](#_Toc121953349)

[Практическая работы №6. Blockchain 51](#_Toc121953350)

[Практическая работа №7. Создание реактивного приложения реального времени с использованием Reactor 57](#_Toc121953351)

[Вывод 63](#_Toc121953352)

# Практическая работа №1: Примитивы синхронизации и их использование при создании клиент серверных приложений

**Ход работы:** для решения первой практической работы были написаны класс Main (Рисунок 1.1), три класса для функций (Рисунки 1.2-1.5), классы для реализации с помощью многопоточности (Рисунок 1.6) и с использованием метода ForkJoin (Рисунок 1.7), класс с используемыми инструментами Tools (Рисунок 1.8).

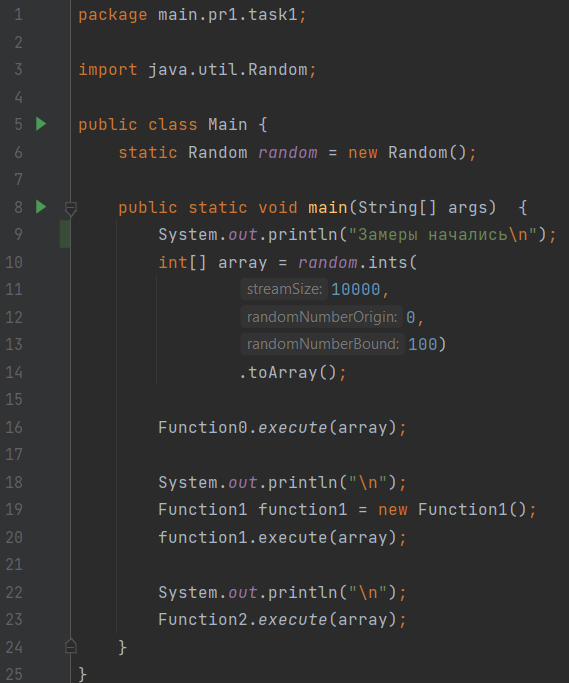


Рисунок 1.1 – Скриншот кода класса Main

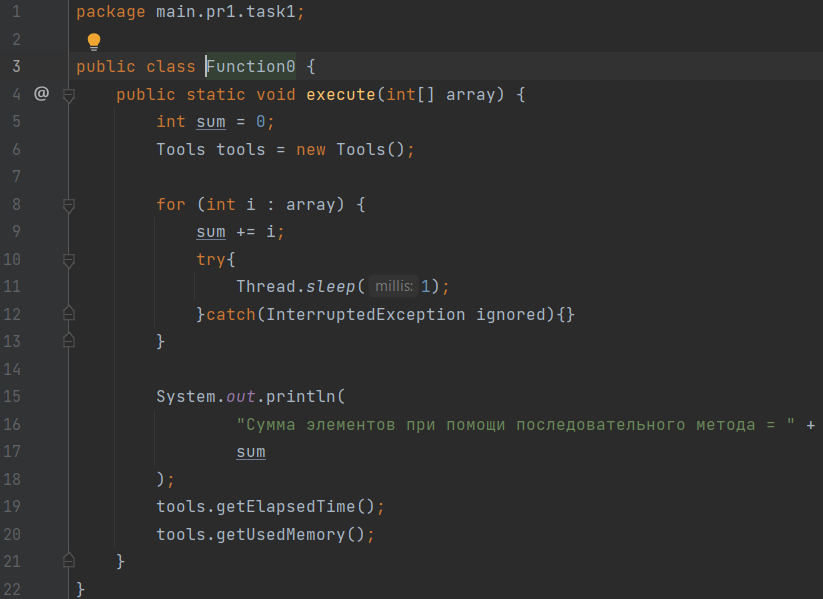


Рисунок 1.2 – Скриншот кода класса императивного решения задачи

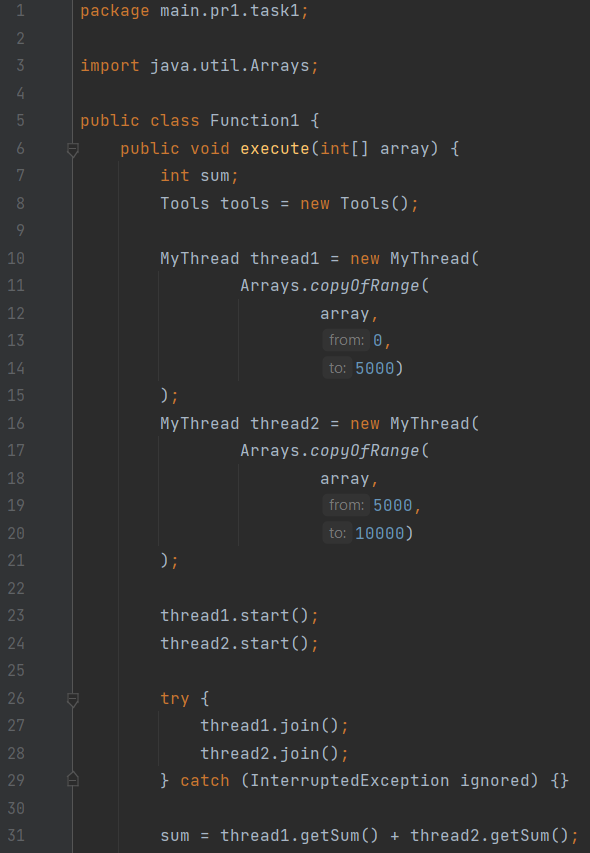


Рисунок 1.3 – Скриншот кода класса для реализации метода многопоточности (Часть 1)

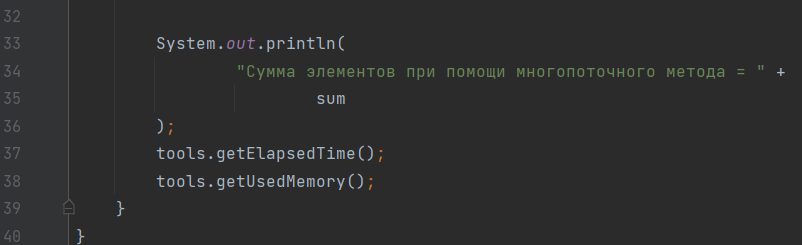


Рисунок 1.4 – Скриншот кода класса для реализации метода многопоточности (Часть 2)

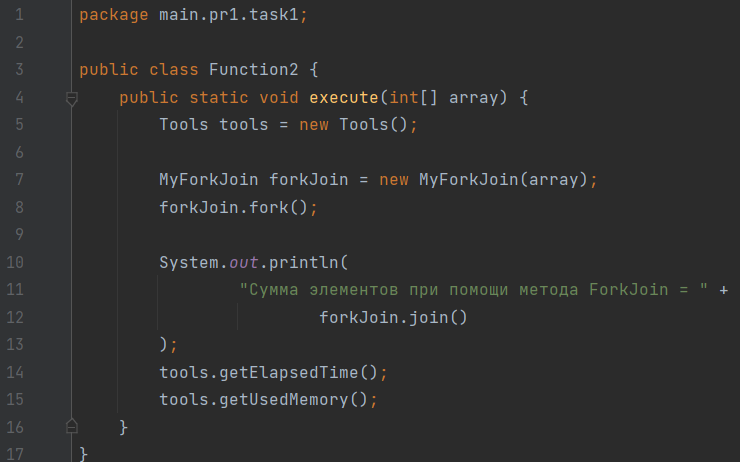


Рисунок 1.5 – Скриншот кода класса для реализации метода ForkJoin



Рисунок 1.6 – Скриншот кода класса наследника Thread

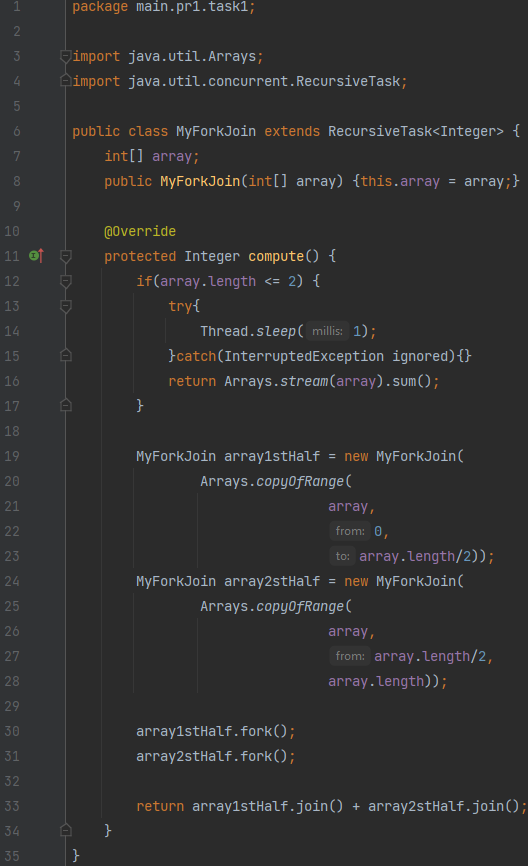


Рисунок 1.7 – Скриншот кода класса наследника для реализации метода ForkJoin



Рисунок 1.8 – Скриншот кода класса вспомогательных методов

После запуска первой программы и ожидания около одной минуты для всех методов выводятся результаты вычислений, что можно увидеть на следующем рисунке (Рисунок 1.9):

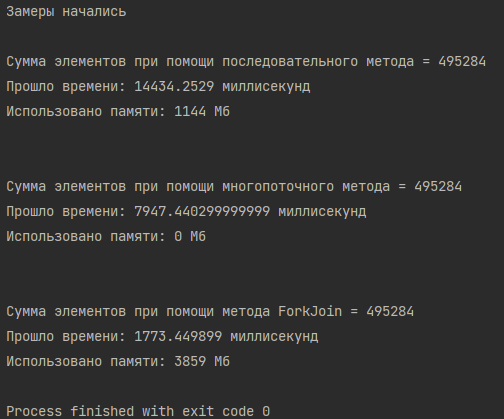


Рисунок 1.9 – Скриншот результата выполнения первой задачи

Затем производился запуск второй программы, в которой вводятся последовательно числа до получения результата вычислений, поскольку вычисления производятся с желаемой задержкой (1-5 секунд), результат выполнения задания представлен на рисунке 1.10.

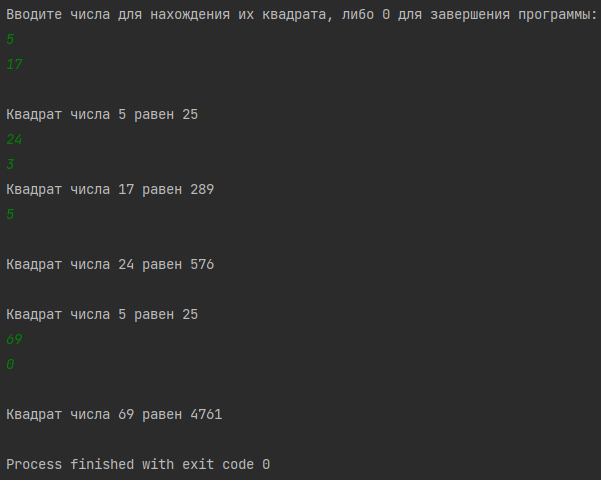


Рисунок 1.10 – Скриншот результата выполнения второй задачи

Наконец, запуск третьей программы производился с целью генерации и дальнейшей обработки ранее сгенерированных файлов в разных потоках, поскольку записей с новых строк больше, чем умещается на скриншоте, результат выполнения третьей задачи представлен на рисунках 1.11-1.12.

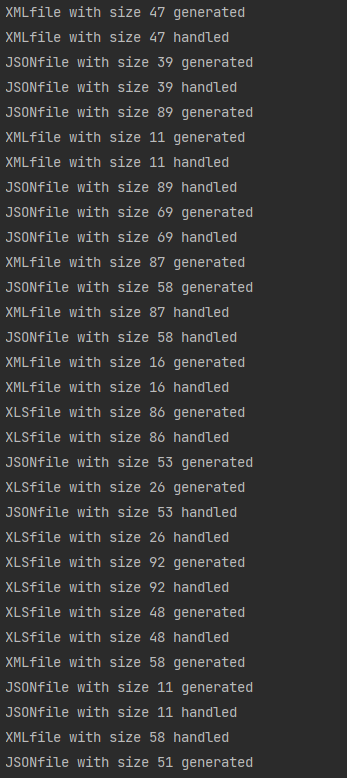


Рисунок 1.11 – Скриншот результата выполнения третьей задачи (Часть 1)

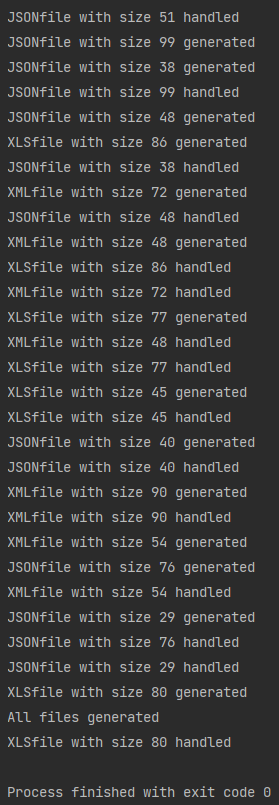


Рисунок 1.12 – Скриншот результата выполнения третьей задачи (Часть 2)

# Практическая работа №2: Java NIO

**Ход работы:** для решения второй практической работы были написаны класс Main (Рисунок 2.1), четыре класса для функций (Рисунки 2.2-2.7), классы для реализации различных методов копирования во втором задании и вспомогательные интерфейс Example и Timer для подсчета времени (Рисунки 2.8-2.13).

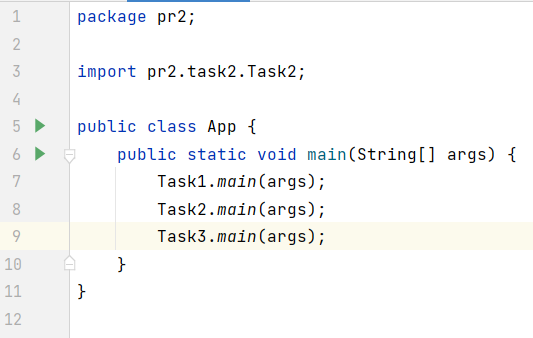


Рисунок 2.1 – Скриншот кода класса Main



Рисунок 2.2 – Скриншот кода класса для решения первого задания



Рисунок 2.3 – Скриншот кода класса для решения второго задания

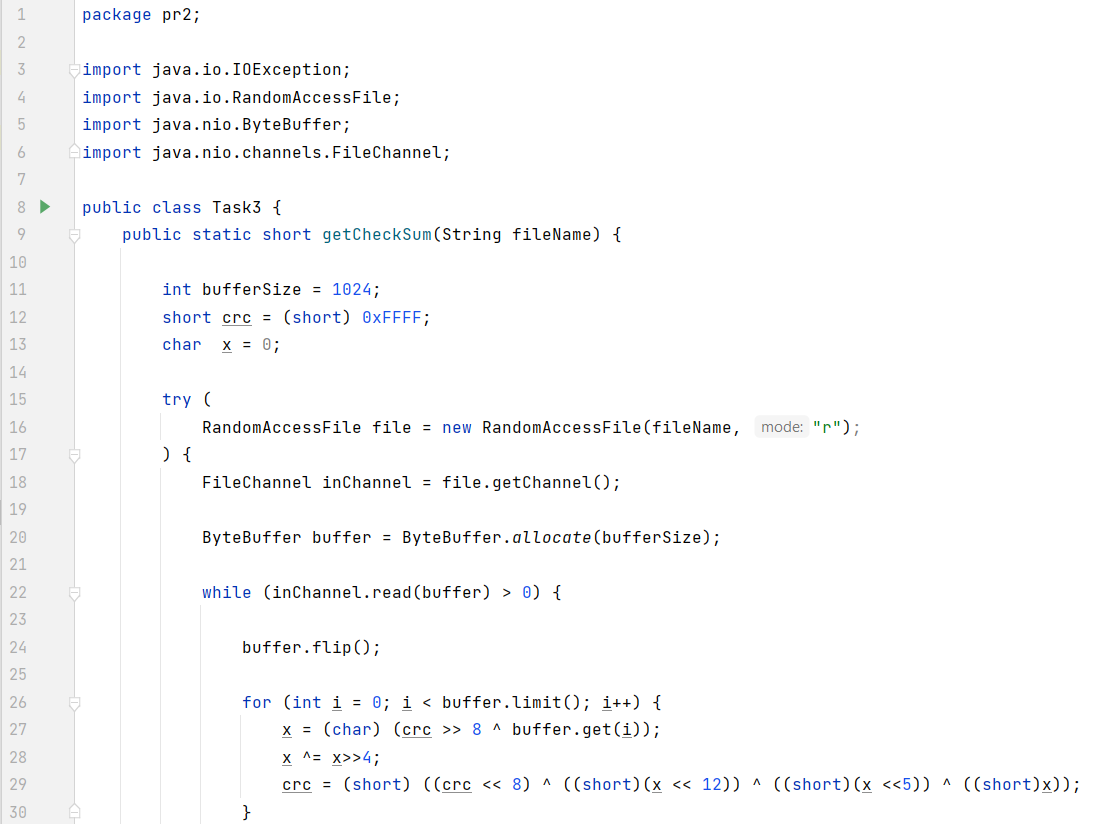


Рисунок 2.4 – Скриншот кода класса для реализации третьей задачи (Часть 1)

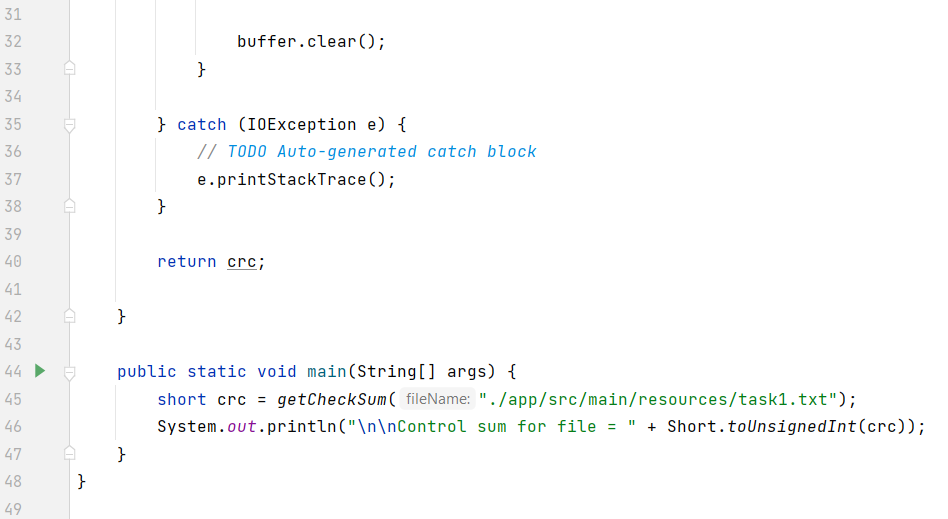


Рисунок 2.5 – Скриншот кода класса для реализации третьей задачи (Часть 2)



Рисунок 2.6 – Скриншот кода класса реализации четвертой задачи (Часть 1)

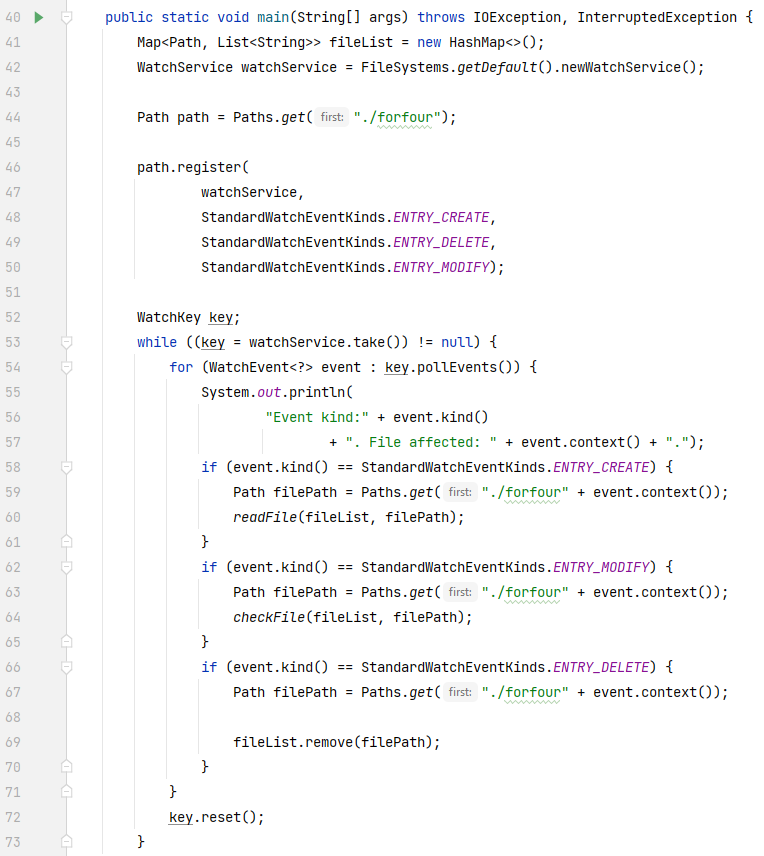


Рисунок 2.7– Скриншот кода класса реализации четвертой задачи (Часть 2)



Рисунок 2.8 – Скриншот кода класса реализации метода Apache Commons IO

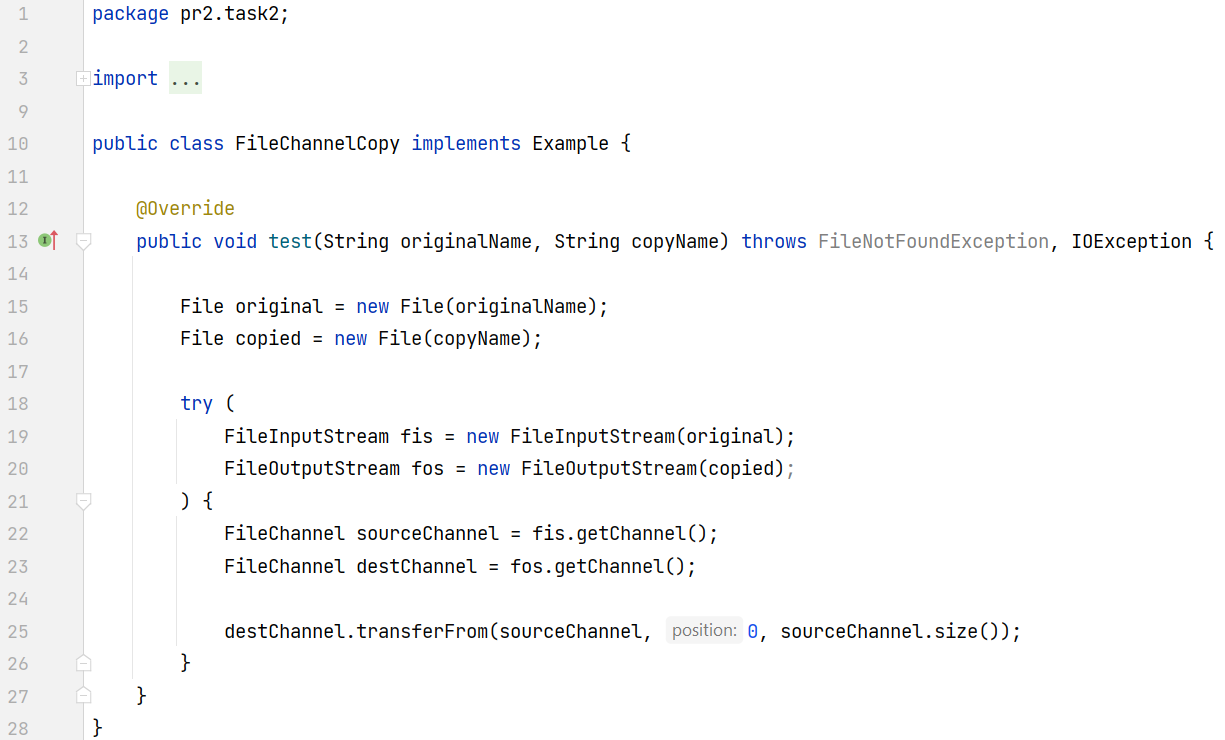


Рисунок 2.9 – Скриншот кода класса реализации метода FileChannel



Рисунок 2.10 – Скриншот кода класса реализации метода File



Рисунок 2.11 – Скриншот кода класса реализации метода FileInputStream/FileOutputStream

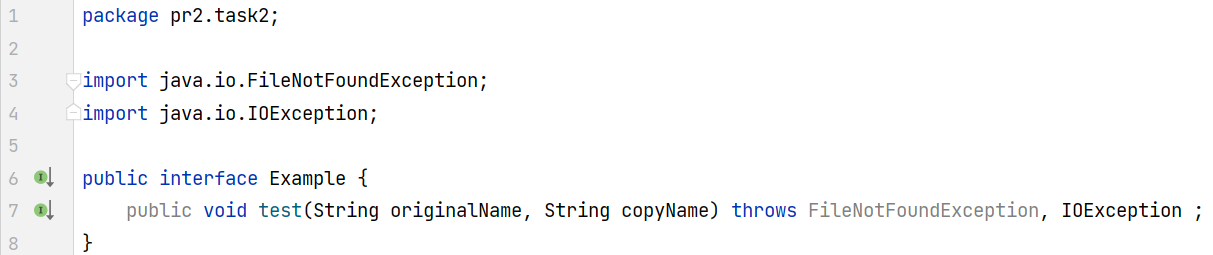


Рисунок 2.12 – Скриншот кода интерфейса Example

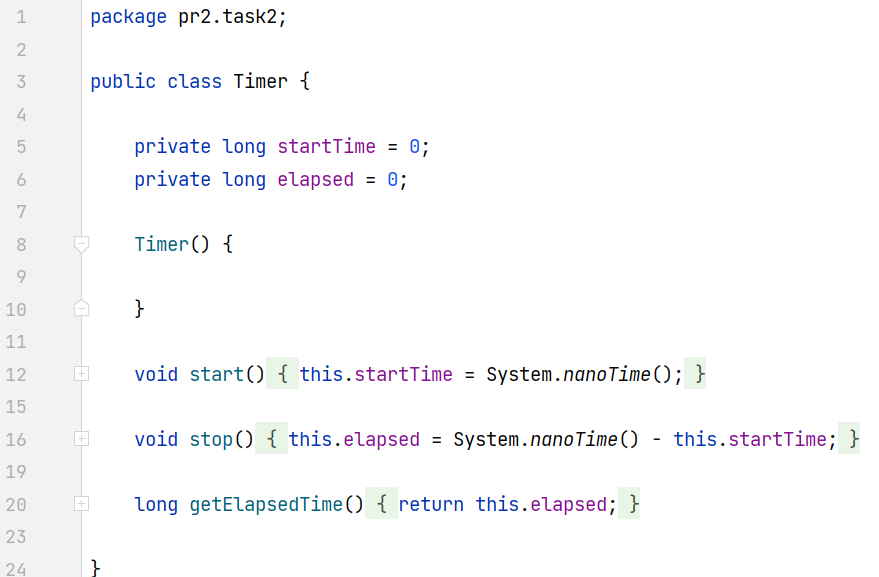


Рисунок 2.13 – Скриншот кода вспомогательного класса Timer

После запуска метода main класса App.java происходит выполнение первого, второго и третьего заданий, что изображено на рисунках 2.14-2.15.

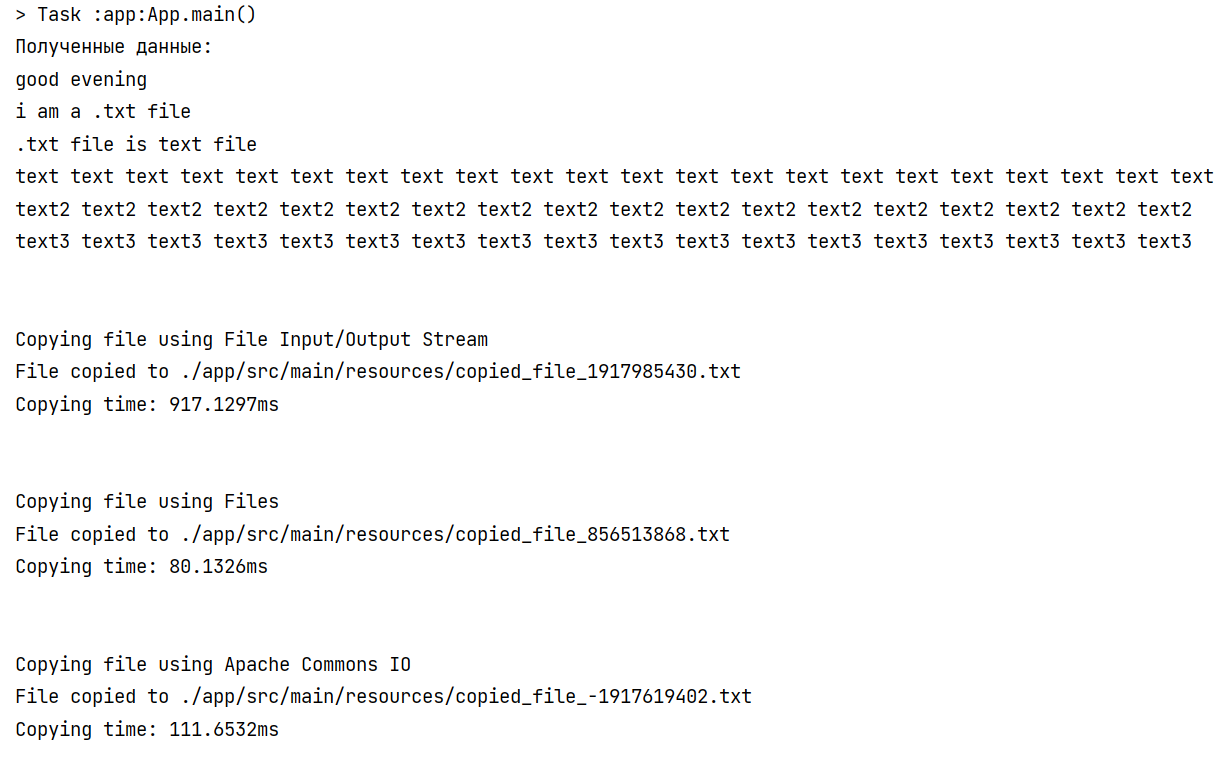


Рисунок 2.14 – Скриншот успешного запуска первого, второго и третьего заданий (Часть 1)

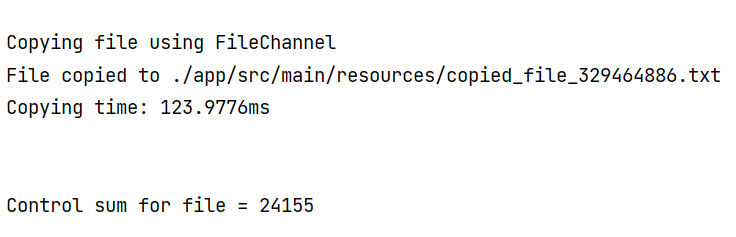


Рисунок 2.15 – Скриншот успешного запуска первого, второго и третьего заданий (Часть 2)

Наконец, после запуска четвертой задачи был создан текстовый документ в соответствующей папке, переименован, изменен и затем удален – все это отражено в консоли, что можно видеть на рисунке 2.16.

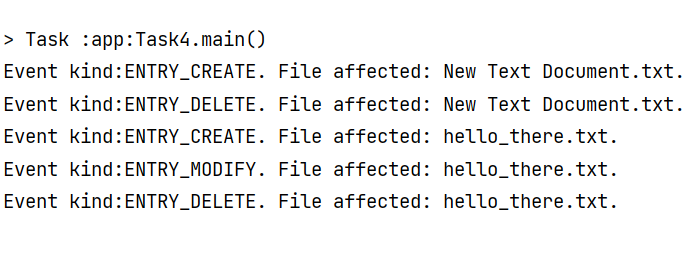


Рисунок 2.16 – Скриншот успешного запуска и работы четвертого задания

# Практическая работа №3. RxJava

**Ход работы:** В первом задании была использована RxJava, что можно видеть на листинге 3.1.

Листинг 3.1 – Код первой задачи

package org.example;  
  
import io.reactivex.rxjava3.core.Observable;  
import io.reactivex.rxjava3.schedulers.Schedulers;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.concurrent.TimeUnit;  
  
public class taskOne {  
 public static int randomNumber(int bot, int top) {  
  
 return (int)Math.*round*(bot + (Math.*random*() \* (top-bot)));  
 }  
 public static String checkResults(int[] array) {  
 if (array[0]>25 && array[1]>70)  
 return "Houston, we have a problem!";  
 if (array[0]>25)  
 return "Bad temperature stats: "+array[0];  
 if (array[1]>70)  
 return "Bad C02 stats: "+array[1];  
 return "We good B^)";  
 }  
 public static void main(String[] args) throws InterruptedException {  
 ExecutorService executor1 = Executors.*newSingleThreadExecutor*();  
 ExecutorService executor2 = Executors.*newSingleThreadExecutor*();  
 Observable<Integer> temp = Observable.*interval*(1000, TimeUnit.*MILLISECONDS*)  
 .subscribeOn(Schedulers.*from*(executor1)).map(a -> *randomNumber*(15,30));  
  
 Observable<Integer> co2 = Observable.*interval*(1000, TimeUnit.*MILLISECONDS*)  
 .subscribeOn(Schedulers.*from*(executor2)).map( a -> *randomNumber*(30, 100));  
  
 Observable.*zip*(temp, co2,  
 (observable1, observable2) -> new int[]{observable1, observable2})  
 .subscribe(item -> System.*out*.println(*checkResults*(item)));  
 }  
}

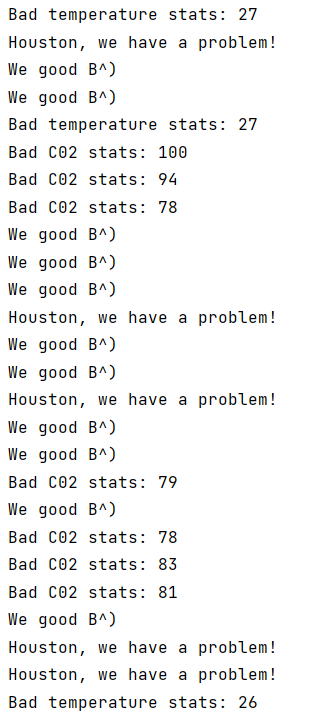


Рисунок 3.1 – Результат работы предыдущего листинга

Для второй практической работы в первом задании была выбрана задача - Преобразовать поток из 1000 случайных чисел от 0 до 1000 в поток, содержащий только числа больше 500, листинг программы представлен ниже.

Листинг 3.2 – Код задания 2.1.2

package org.example;  
import io.reactivex.rxjava3.core.Flowable;  
  
import java.util.Arrays;  
  
public class taskTwoOne {  
 public static Integer[] fillArray() {  
 Integer[] array = new Integer[1000];  
 for (int i = 0; i < array.length; i++) {  
 array[i] = (int) Math.*round*((Math.*random*() \* 1000));  
 }  
 return array;  
 }  
  
 public static void main(String[] args) {  
 Integer[] array = *fillArray*();  
 System.*out*.println(Arrays.*toString*(array));  
 Flowable<Integer> flow = Flowable.*fromArray*(array);  
 flow.subscribe(num -> System.*out*.print(num+" "));  
 System.*out*.print("\n");  
 Flowable<Integer> next = flow.filter(item -> item > 500).map(item -> item\*item);  
 next.subscribe(num -> System.*out*.print(num+" "));  
 }  
}

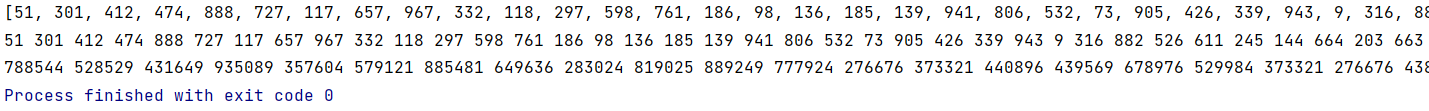


Рисунок 3.2 – Результат работы предыдущего листинга

Для второй практической работы во втором задании была выбрана задача - 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), листинг программы представлен ниже.

Листинг 3.3 – Код задания 2.2.1

package org.example;  
import io.reactivex.rxjava3.core.Flowable;  
  
import java.util.Random;  
  
public class taskTwoTwo {  
 public static Character[] randomFiller(String digits, Random random) {  
 Character[] array = new Character[1000];  
 for (int i = 0; i < array.length; i++) {  
 array[i] = digits.charAt(random.nextInt(digits.length()));  
 }  
 return array;  
 }  
  
  
 public static void main(String[] args) {  
 Random random = new Random();  
 final String letters = "ABCDEFGHIGKLM";  
 final String digits2 = "3567893200123";  
 Flowable<Character> firstFlow = Flowable.*fromArray*(*randomFiller*(letters, random));  
 Flowable<Character> secondFlow = Flowable.*fromArray*(*randomFiller*(digits2, random));  
 Flowable<String> finalFlow = Flowable.*zip*(firstFlow, secondFlow,  
 (char1, char2) -> String.*valueOf*(char1)+char2);  
 finalFlow.subscribe(item -> System.*out*.print(item+" "));  
 }  
}

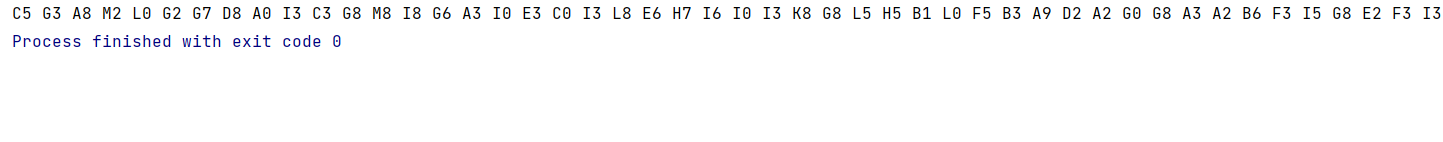


Рисунок 3.3 – Результат работы предыдущего листинга

Для второй практической работы во третьем задании была выбрана задача - Дан поток из 10 случайных чисел. Сформировать поток, содержащий только первые 5 чисел, листинг программы представлен ниже.

Листинг 3.4 – Код задания 2.3.2

package org.example;  
  
  
import io.reactivex.rxjava3.core.Flowable;  
import rx.Observable;  
  
import java.util.Arrays;  
import java.util.List;  
  
public class taskTwoThree {  
 public static void main(String[] args) {  
 List<Integer> numbers = Arrays.*asList*(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);  
 Observable.*from*(numbers)  
 .takeWhile(number -> number < 6)  
 .subscribe(System.*out*::println);  
 }  
}

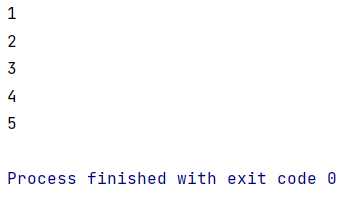


Рисунок 3.4 – Результат работы предыдущего листинга

В третьем задании был реализован класс UserFriend и поля — int userId, friendId, листинг предоставлен ниже.

Листинг 3.5 – Код задания 3

package org.example;  
  
  
import io.reactivex.rxjava3.core.Observable;  
  
public class taskThree {  
 public static int randomNumber(int bot, int top) {  
 return (int)Math.*round*(bot + (Math.*random*() \* (top-bot)));  
 }  
 public static UserFriend[] fillArray() {  
 UserFriend[] array = new UserFriend[1000];  
 for (int i = 0; i < array.length; i++) {  
 int userId = (int) Math.*round*((Math.*random*() \* 50));  
 int friendId = (int) Math.*round*((Math.*random*() \* 50));  
 array[i] = new UserFriend(userId,friendId);  
 }  
 return array;  
 }  
 public static 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; }  
 @Override  
 public String toString() { return "UserFriend{" + "userId=" + userId + ", friendId=" + friendId + "}\n"; }  
 }  
 public static Observable<UserFriend> getFriends(int userId, Observable<UserFriend> users) {  
 return users.filter(x -> x.getUserId() == userId);  
 }  
  
 public static void main(String[] args) {  
 UserFriend[] userFriends = *fillArray*();  
 Integer[] randIds = new Integer[15];  
 for (int i=0; i<randIds.length; i++)  
 randIds[i] = *randomNumber*(0,50);  
 Observable<UserFriend> userFriendObservable = Observable.*fromArray*(userFriends);  
 Observable<UserFriend> filtered = Observable.*empty*();  
 for (Integer randId : randIds) {  
 filtered = filtered.concatWith(*getFriends*(randId, userFriendObservable));  
 }  
 filtered.subscribe(item -> System.*out*.print(item.toString()+" "));  
 }  
}



Рисунок 3.5 – Результат работы предыдущего листинга

В четвертом задании был реализована система:

Файл. Имеет следующие характеристики:

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

1. Размер файла — целочисленное значение от 10 до 100.

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

Код задания представлен на листинге 3.6.

Листинг 3.6 – Код задания 4

package org.example;  
  
import rx.Observable;  
import rx.Observer;  
  
public class Task4 {  
 private int count = 0;  
  
 public class Handler implements Runnable{  
 Task4File file;  
 Handler(Task4File file){  
 this.file = file;  
 }  
  
 @Override  
 public void run() {  
 try {  
 Thread.*sleep*(file.getSize()\* 70L);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 count--;  
 System.*out*.println(file.getType() + " file with size " + file.getSize() + " bytes processed");  
 }  
 }  
  
 Observer<Task4File> getQueue() {  
 return new Observer<>() {  
 @Override  
 public void onCompleted() {  
 }  
  
 @Override  
 public void onError(Throwable throwable) {  
 }  
  
 @Override  
 public void onNext(Task4File file) {  
 Handler handler = new Handler(file);  
 new Thread(handler).start();  
 }  
 };  
 }

Продолжение листинга 3.6 представлено ниже:

public Observable<Task4File> getGenerator(){  
 String[] types = new String[]{"xml", "json", "xls"};  
 return Observable.*create*(subscriber -> {  
 Runnable r = () -> {  
 while (true){  
 String type = types[(int)(Math.*random*()\*3)];  
 int size = (int)(Math.*random*()\*90)+10;  
 try {  
 Thread.*sleep*((int)(Math.*random*()\*900)+100);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 Task4File file = new Task4File(type, size);  
 System.*out*.println(type + " file with size " + size + " bytes generated");  
 while (count == 4){  
 try {  
 Thread.*sleep*(1);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 System.*out*.println(file.getType() + " file with size " +  
 file.getSize() + " bytes added to queue");  
 count++;  
 subscriber.onNext(file);  
 }  
 };  
 new Thread(r).start();  
 }  
 );  
 }  
  
 public static void main(String[] args){  
 Task4 task4 = new Task4();  
 Observer<Task4File> queue = task4.getQueue();  
 Observable<Task4File> generator = task4.getGenerator();  
 generator.subscribe(queue);  
 }  
}

Листинг 3.7 – Код файла Task4File

package org.example;  
  
public record Task4File(String type, int size) {  
 public String getType() {  
 return type;  
 }  
  
 public int getSize() {  
 return size;  
 }  
  
}



Рисунок 3.6 – Результат работы предыдущего листинга

# Практическая работа №4. Rsocket

**Ход работы:**

Была реализована клиент-серверная система, основные элементы которой представлены на следующих скриншотах.

В первую очередь была разработана база данных на базе PosgreSQL. На рисунках 4.1-4.2 представлены скриншоты содержимого баз данных.

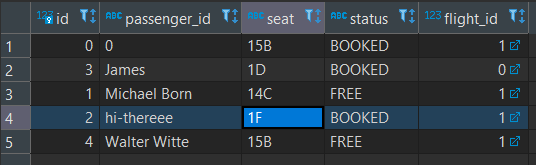


Рисунок 4.1 – Скриншот содержимого базы данных с билетами

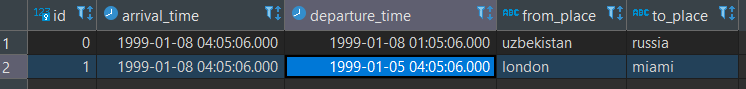


Рисунок 4.2 – Скриншот содержимого базы данных с полетами

Развертывание базы данных происходило в контейнере, docker-compose файл которого представлен на рисунке 4.3.

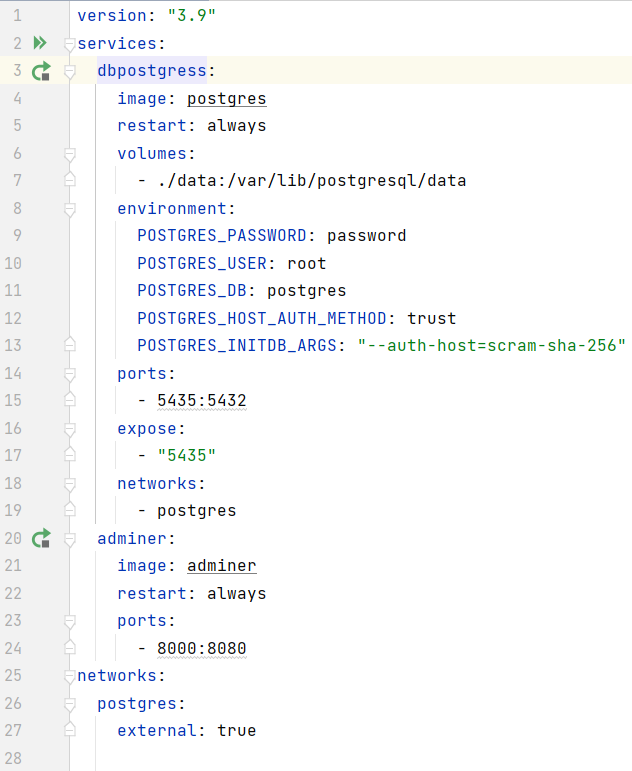


Рисунок 4.3 – Скриншот содержимого файла docker-compose.yaml

Для успешной сборки проекта использовался сборщик maven, на рисунках 4.4-4.5 представлены зависимости в проекте.

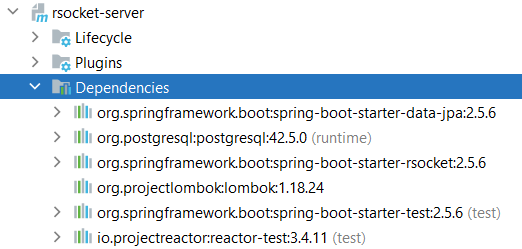


Рисунок 4.4 – Скриншот зависимостей серверной части

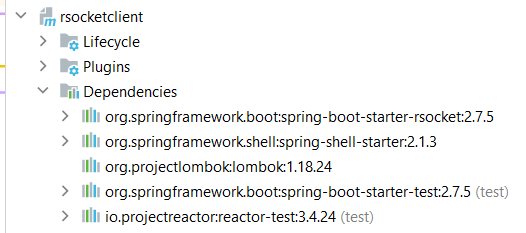


Рисунок 4.5 – Скриншот зависимостей клиентской части

В результате сборки и запуска приложения, были реализованы следующие методы, требуемые в постановке задачи:

Request-Response реализован с помощью команды *book --ticket-id --passenger-id*, благодаря ей происходит бронь места. Пример работы представлен на рисунках 4.6-4.7.

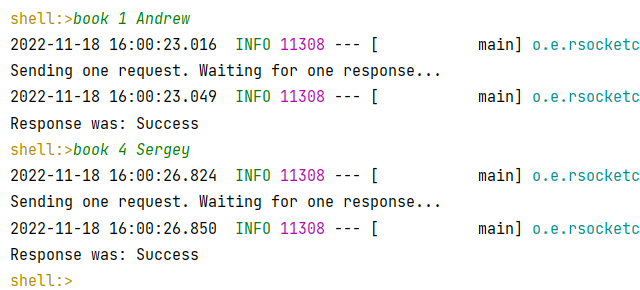


Рисунок 4.6 – Скриншот реализации метода Request-Response

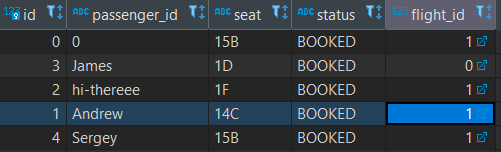


Рисунок 4.7 – Скриншот результата использования метода

Request-Stream метод реализован посредством команды *flights*, благодаря которой происходит вывод всех рейсов. На рисунке 4.8 представлены скриншоты действия метода.

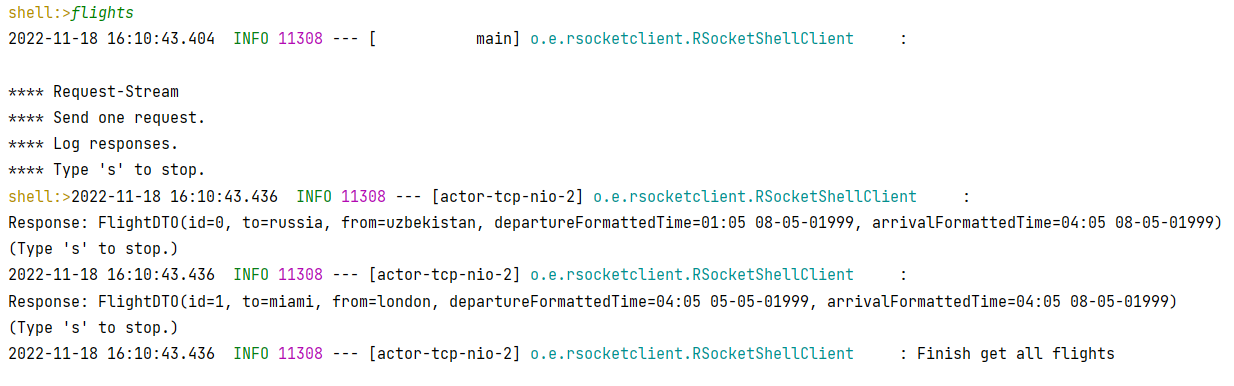


Рисунок 4.8 – Скриншот реализации метода Request-Stream

Channel метод реализован с помощью команды *tickets --flight-ids*, который производит вывод броней рейсов по идентификатору полета. На рисунке 4.9 изображены скриншоты работы метода.

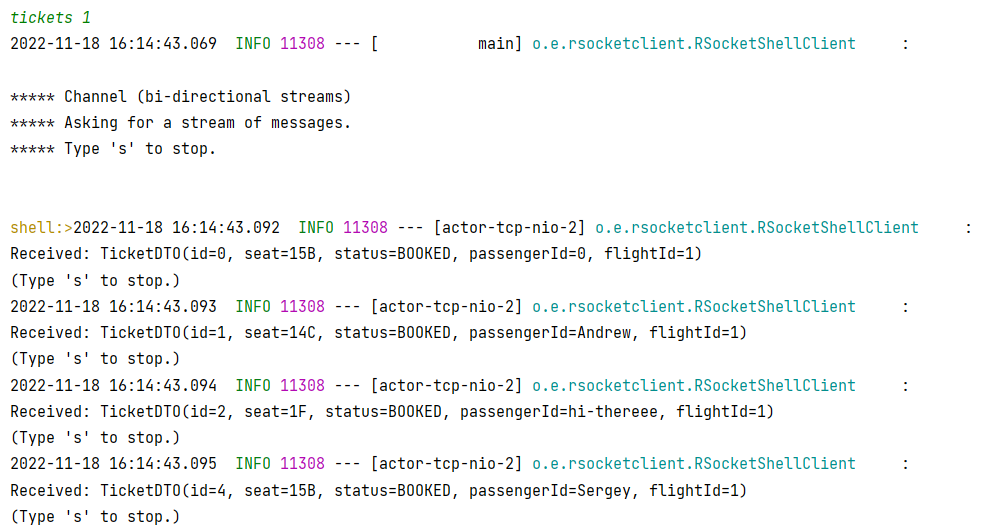


Рисунок 4.9 – Скриншот реализации и использования метода Channel

Fire-and-Forget метод реализован посредством команды *delete-reservation --ticket-id*. С ее помощью происходит удаление брони на место. На следующих двух рисунках (4.11-4.13) изображены скриншоты работы метода.

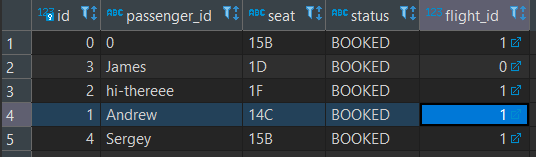


Рисунок 4.11 – Скриншот содержимого базы данных до выполнения метода



Рисунок 4.12 – Скриншот реализации метода Fire-and-Forget

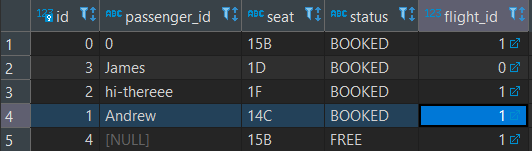


Рисунок 4.13 – Скриншот содержимого базы данных после выполнения метода

# Практическая работа №5. Decentralized Distributed System

Разработать децентрализованную распределенную систему для хранения файлов с использованием Spring/Spring Boot.

Требуемый функционал системы: загрузка и выгрузка файлов различных форматов.

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.

Код программы представлен на Листингах 5.1-5.10. Результат выполнения работы представлен на Рисунках 5.1-5.6.

Листинг 5.1 – Файл application.properties

spring.servlet.multipart.enabled=true  
spring.servlet.multipart.max-file-size=10MB  
spring.servlet.multipart.max-request-size=15MB  
  
spring.main.lazy-initialization=true  
  
spring.dataSource.url=jdbc:postgresql://192.168.19.153:5432/postgres  
spring.dataSource.username=postgres  
spring.dataSource.password=shel  
  
spring.jpa.database=postgresql  
spring.datasource.driver-class-name=org.postgresql.Driver  
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect

Листинг 5.2 — Файл init.sql

create table if not exists public.binary\_file  
(  
 id serial primary key,  
 file\_name varchar,  
 file\_data bytea  
);

Листинг 5.3 — Файл deployment.yaml

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 labels:  
 app: distributed-system  
 name: distributed-system  
 namespace: default  
spec:  
 replicas: 4  
 strategy:  
 type: RollingUpdate  
 rollingUpdate:  
 maxSurge: 3  
 maxUnavailable: 1  
 selector:  
 matchLabels:  
 app: distributed-system  
 template:  
 metadata:  
 labels:  
 app: distributed-system  
 spec:  
 containers:  
 - name: distributed-system  
 image: pr5  
 imagePullPolicy: Never  
 ports:  
 - containerPort: 8080

Листинг 5.4 — Файл ingress.yaml

apiVersion: networking.k8s.io/v1  
kind: Ingress  
metadata:  
 name: distributed-system-ingress  
spec:  
 rules:  
 - http:  
 paths:  
 - path: /download  
 pathType: Prefix  
 backend:  
 service:  
 name: distributed-system  
 port:  
 number: 8080  
 - path: /upload  
 pathType: Prefix  
 backend:  
 service:  
 name: distributed-system  
 port:  
 number: 8080

}  
}

Листинг 5.5 — Файл service.yaml

apiVersion: v1  
kind: Service  
metadata:  
 labels:  
 app: distributed-system  
 name: distributed-system  
 namespace: default  
spec:  
 selector:  
 app: distributed-system  
 ports:  
 - protocol: TCP  
 port: 8080  
 targetPort: 8080  
 type: LoadBalancer

Листинг 5.6 — Класс FileController

package ru.mirea.practice5.controllers;  
  
import java.io.IOException;  
import java.util.Objects;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpHeaders;  
import org.springframework.http.MediaType;  
import org.springframework.http.ResponseEntity;  
import org.springframework.util.StringUtils;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
import org.springframework.web.bind.annotation.RestController;  
import org.springframework.web.multipart.MultipartFile;  
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;  
import ru.mirea.practice5.services.FileService;  
  
@RestController  
public class FileController {  
 @Autowired  
 FileService service;  
  
 @PostMapping("/upload")  
 public ResponseEntity uploadToDB(@RequestParam("file") MultipartFile file) throws IOException {  
 String fileName = StringUtils.*cleanPath*(Objects.*requireNonNull*(file.getOriginalFilename()));  
 service.uploadFile(StringUtils.*cleanPath*(file.getOriginalFilename()), file.getBytes());  
 String fileDownloadUri = ServletUriComponentsBuilder.*fromCurrentContextPath*()  
 .path("/download/")  
 .path(fileName)  
 .toUriString();  
 return ResponseEntity.*ok*(fileDownloadUri); }  
  
 @GetMapping("/download/{fileName:.+}")  
 public ResponseEntity downloadFromDB(@PathVariable String fileName) {

Продолжение Листинга 5.6

String mediaType = service.getMediaType(fileName);

return ResponseEntity.*ok*()  
 .contentType(MediaType.*parseMediaType*(mediaType))  
 .header(HttpHeaders.*CONTENT\_DISPOSITION*, "attachment; filename=\"" + fileName + "\"")  
 .body(service.downloadFile(fileName)); }}

Листинг 5.7 – Класс BinaryFile

package ru.mirea.practice5.models;  
  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.Table;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Entity  
@Table(name = "binary\_file")  
@Data  
@NoArgsConstructor  
public class BinaryFile {  
 @Id  
 @GeneratedValue(strategy= GenerationType.*AUTO*)  
 private int id;  
  
 @Column(name = "file\_name")  
 String fileName;  
  
 @Column(name = "file\_data")  
 byte[] fileData;  
  
 public BinaryFile(String name, byte[] data) {  
 this.fileData = data;  
 this.fileName = name;  
 }  
}

Листинг 5.8 – Класс BinaryFileRepository

package ru.mirea.practice5.repositories;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import ru.mirea.practice5.models.BinaryFile;  
  
public interface BinaryFileRepository extends JpaRepository<BinaryFile, Integer> {  
 BinaryFile findByFileName(String fileName);  
  
 Boolean existsByFileName(String fileName);  
}

Листинг 5.9 — Класс Practice5Application

package ru.mirea.practice5;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class Practice5Application {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(Practice5Application.class, args);  
 }  
}

Листинг 5.10 — Класс FileService

package ru.mirea.practice5.services;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.MediaType;  
import org.springframework.stereotype.Service;  
import ru.mirea.practice5.models.BinaryFile;  
import ru.mirea.practice5.repositories.BinaryFileRepository;  
  
@Service  
public class FileService {  
  
 @Autowired  
 BinaryFileRepository repository;  
  
 public void uploadFile(String name, byte[] data) {  
 if (!repository.existsByFileName(name)) {  
 BinaryFile file = new BinaryFile(name, data);  
 repository.save(file);  
 }  
 }  
  
 public byte[] downloadFile(String name) {  
 BinaryFile binaryFile = repository.findByFileName(name);  
 return binaryFile.getFileData();  
 }  
  
 public String getMediaType(String name) {  
 String getFormat = name.split("\\.")[1];  
 String type = null;  
 switch (getFormat) {  
 case "pdf" :  
 type = MediaType.*APPLICATION\_PDF\_VALUE*;  
 break;  
 case "txt" :  
 type = MediaType.*TEXT\_PLAIN\_VALUE*;  
 break;  
 case "jpg", "jpeg" :  
 type = MediaType.*IMAGE\_JPEG\_VALUE*;  
 break;  
 case "png" :  
 type = MediaType.*IMAGE\_PNG\_VALUE*;  
 break;  
 }  
 return type;  
 }  
  
}

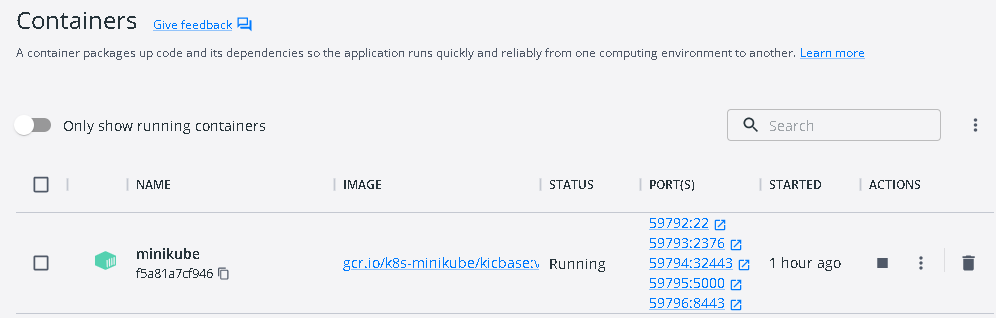
****

Рисунок 5.1 — Развернутый контейнер minikube

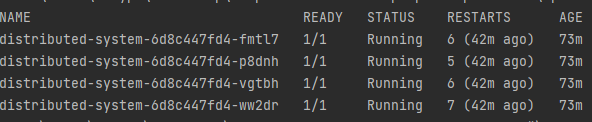


Рисунок 5.2 — Запущенные сервера

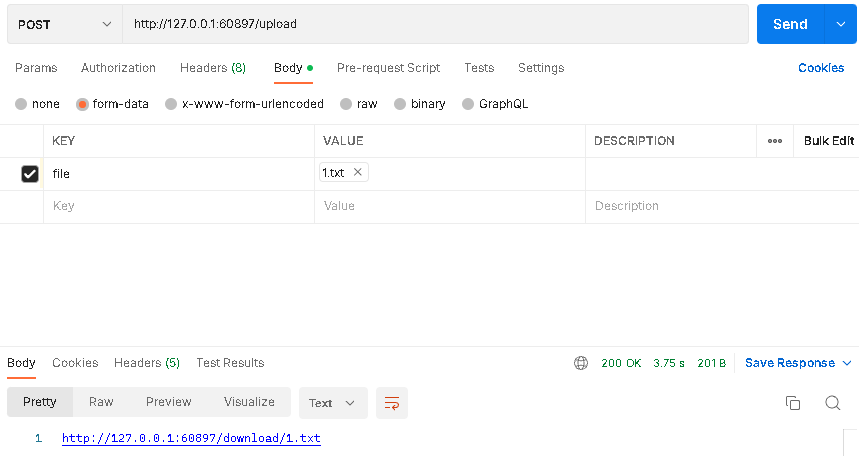


Рисунок 5.3 – Успешная отправка файла при запущенных серверах

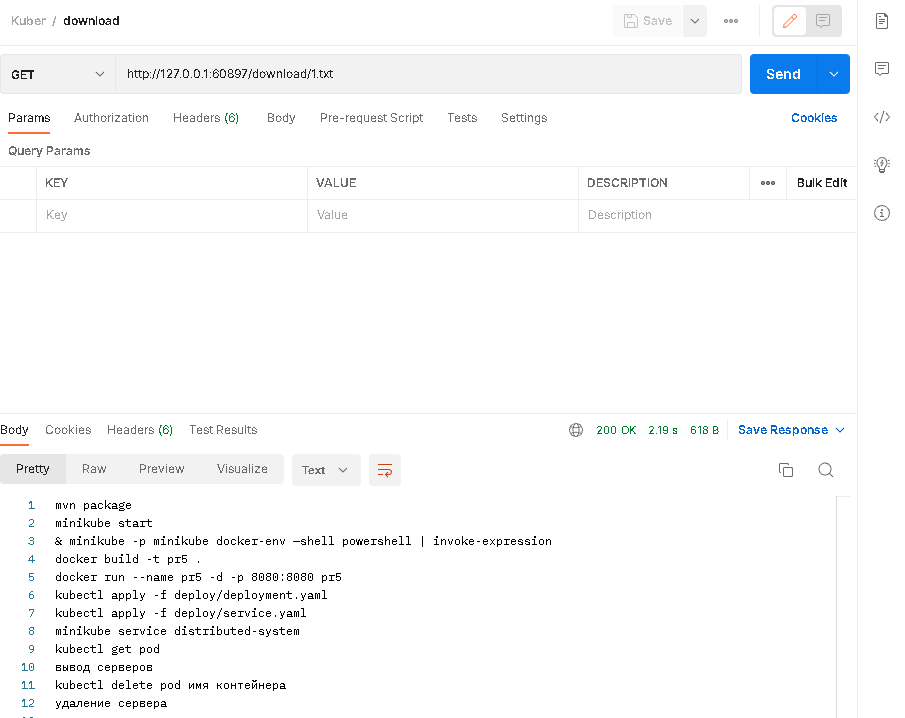


Рисунок 5.4 – Успешное получение файла с запущенными серверами



Рисунок 5.5 — Удаленный сервер

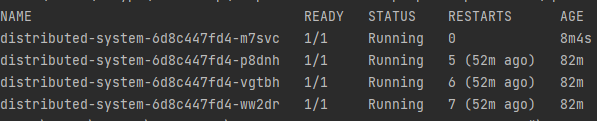


Рисунок 5.6 — Запущенные сервера с одним новым

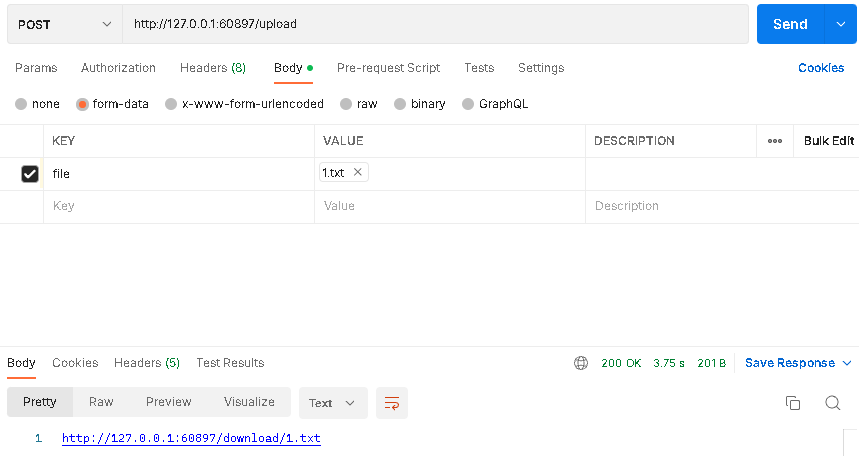


Рисунок 5.7 — Успешная отправка файла с удаленными серверами

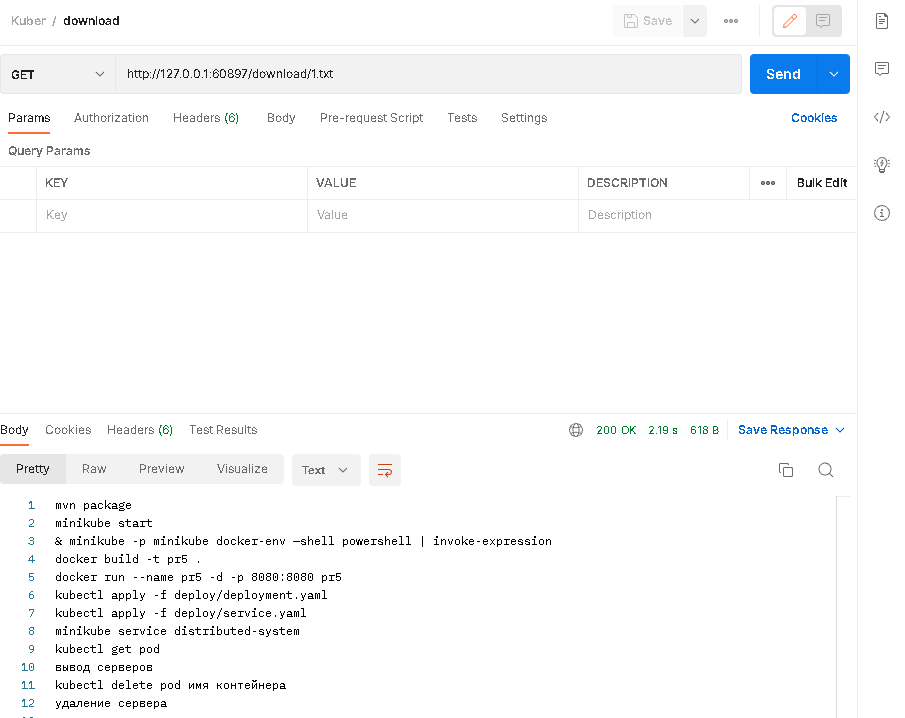


Рисунок 5.8 — Успешное получение файла с удаленными серверами

# Практическая работы №6. Blockchain

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

2) Написать unit-тесты.

3) Продемонстрировать работу через встроенные инструменты Remix IDE.

В ходе выполнения настоящей практической работы проведено исследование блокчейн технологий и технологий смарт-контрактов. Для реализации использован язык Solidity по индивидуальному варианту – публикация заданий с вознаграждением.

Код программы представлен на Листинге 6.1. Результат выполнения работы представлен на Рисунках 6.1-6.4.

Листинг 6.1 – Файл Bounties-complete.sol

pragma solidity ^0.5.0;

contract Bounties {

  enum BountyStatus { CREATED, ACCEPTED, CANCELLED }

  Bounty[] public bounties;

  mapping(uint=>Fulfillment[]) fulfillments;

  struct Bounty {

      address payable issuer;

      uint deadline;

      string data;

      BountyStatus status;

      uint amount; //in wei

  }

  struct Fulfillment {

      bool accepted;

      address payable fulfiller;

      string data;

  }

constructor() public {}

function issueBounty(

Продолжение Листинга 6.1

string calldata \_data,

      uint64 \_deadline

  )

      external

      payable

      hasValue()

      validateDeadline(\_deadline)

      returns (uint)

  {

      bounties.push(Bounty(msg.sender, \_deadline, \_data, BountyStatus.CREATED, msg.value));

      emit BountyIssued(bounties.length - 1,msg.sender, msg.value, \_data);

      return (bounties.length - 1);

  }

  function fulfillBounty(uint \_bountyId, string memory \_data)

    public

    bountyExists(\_bountyId)

    notIssuer(\_bountyId)

    hasStatus(\_bountyId, BountyStatus.CREATED)

isBeforeDeadline(\_bountyId)

  {

    fulfillments[\_bountyId].push(Fulfillment(false, msg.sender, \_data));

    emit BountyFulfilled(\_bountyId, msg.sender, (fulfillments[\_bountyId].length - 1),\_data);

}

  function acceptFulfillment(uint \_bountyId, uint \_fulfillmentId)

      public

      bountyExists(\_bountyId)

      fulfillmentExists(\_bountyId,\_fulfillmentId)

      onlyIssuer(\_bountyId)

      hasStatus(\_bountyId, BountyStatus.CREATED)

      fulfillmentNotYetAccepted(\_bountyId, \_fulfillmentId)

  {

      fulfillments[\_bountyId][\_fulfillmentId].accepted = true;

      bounties[\_bountyId].status = BountyStatus.ACCEPTED;

      fulfillments[\_bountyId][\_fulfillmentId].fulfiller.transfer(bounties[\_bountyId].amount);

      emit FulfillmentAccepted(\_bountyId, bounties[\_bountyId].issuer, fulfillments[\_bountyId][\_fulfillmentId].fulfiller, \_fulfillmentId, bounties[\_bountyId].amount);

  }

Продолжение Листинга 6.1

  function cancelBounty(uint \_bountyId)

      public

      bountyExists(\_bountyId)

      onlyIssuer(\_bountyId)

      hasStatus(\_bountyId, BountyStatus.CREATED)

  {

      bounties[\_bountyId].status = BountyStatus.CANCELLED;

      bounties[\_bountyId].issuer.transfer(bounties[\_bountyId].amount);

      emit BountyCancelled(\_bountyId, msg.sender, bounties[\_bountyId].amount);

  }

  modifier hasValue() {

      require(msg.value > 0);

      \_;

  }

  modifier bountyExists(uint \_bountyId){

    require(\_bountyId < bounties.length);

    \_;

  }

  modifier fulfillmentExists(uint \_bountyId, uint \_fulfillmentId){

    require(\_fulfillmentId < fulfillments[\_bountyId].length);

    \_;

  }

  modifier hasStatus(uint \_bountyId, BountyStatus \_desiredStatus) {

    require(bounties[\_bountyId].status == \_desiredStatus);

    \_;

  }

  modifier onlyIssuer(uint \_bountyId) {

      require(msg.sender == bounties[\_bountyId].issuer);

      \_;

  }

modifier notIssuer(uint \_bountyId) {

      require(msg.sender != bounties[\_bountyId].issuer);

      \_;

  }

  modifier fulfillmentNotYetAccepted(uint \_bountyId, uint \_fulfillmentId) {

    require(fulfillments[\_bountyId][\_fulfillmentId].accepted == false);

    \_;

  }

  modifier validateDeadline(uint \_newDeadline) {

      require(\_newDeadline > now);

      \_;

  }

  modifier isBeforeDeadline(uint \_bountyId) {

    require(now < bounties[\_bountyId].deadline);

    \_;

Продолжение Листинга 6.1

event BountyIssued(uint bounty\_id, address issuer, uint amount, string data);

  event BountyFulfilled(uint bounty\_id, address fulfiller, uint fulfillment\_id, string data);

  event FulfillmentAccepted(uint bounty\_id, address issuer, address fulfiller, uint indexed fulfillment\_id, uint amount);

  event BountyCancelled(uint indexed bounty\_id, address indexed issuer, uint amount);

}

В представленном варианте разработан следующий функционал:

• issueBounty() - создаёт задание (хранит в себе описание работ и срок выполнения).

• fulfilBounty(uint \_bountyId, string \_data) – эта функция должна сохранить запись о выполнении задачи с заданным идентификатором-индексом. msg.sender следует трактовать как учетную запись исполнителя.

• acceptFulfilment(uint \_bountyId, uint \_fulfilmentId) – вызвав эту функцию, создатель контракта подтверждает приемку работы и соглашается, что она была выполнена надлежащим образом. При этом вознаграждение выплачивается исполнителю при условии, что существует заявка от исполнителя.

• cancelBounty(uint \_bountyId) – эта функция должна отменить заказ на работу при условии отсутствия факта приёмки этого заказа. В этом случае средства возвращаются создателю задания

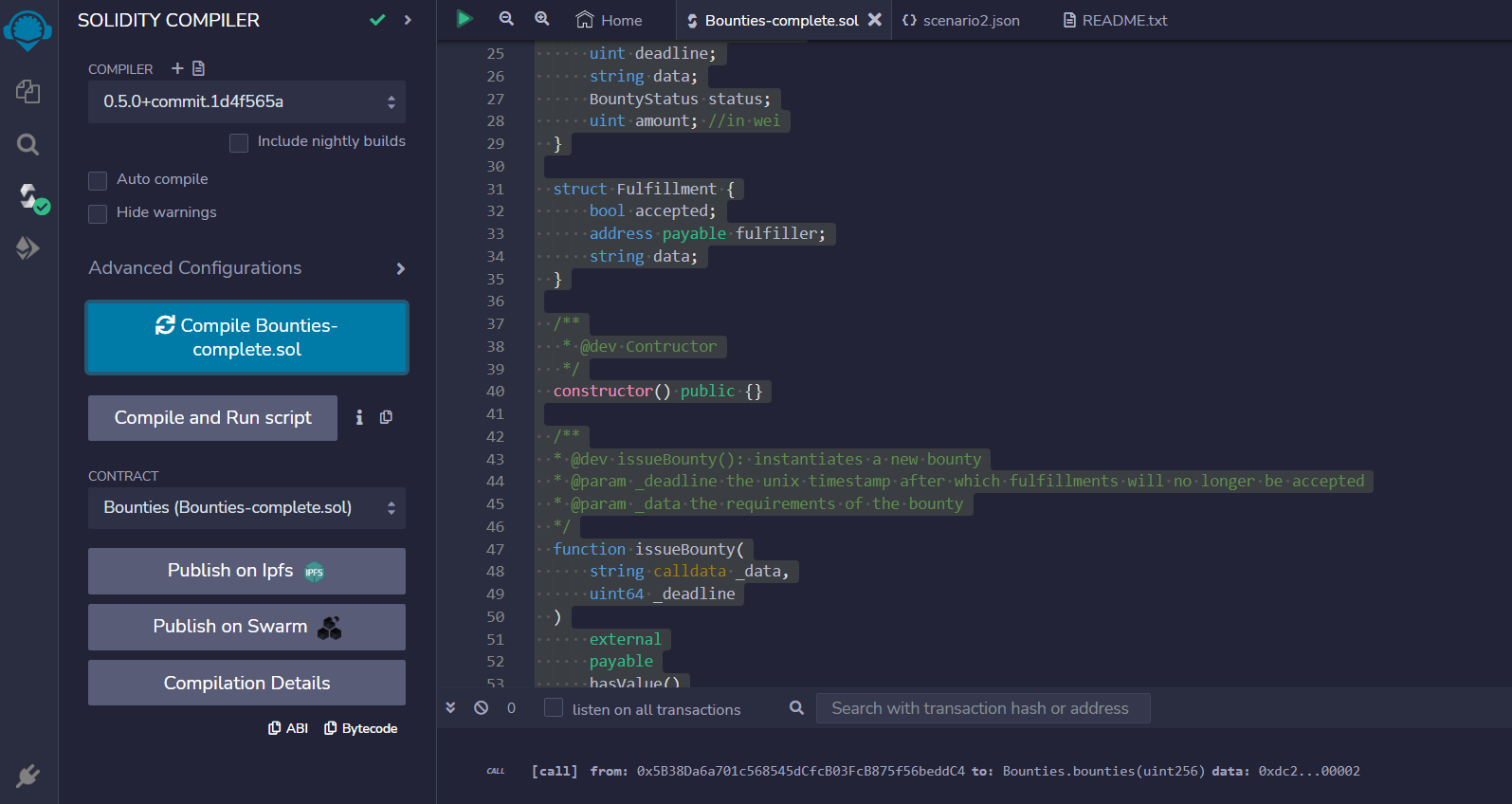


Рисунок 6.1 — Тестирование компиляции смарт-контракта

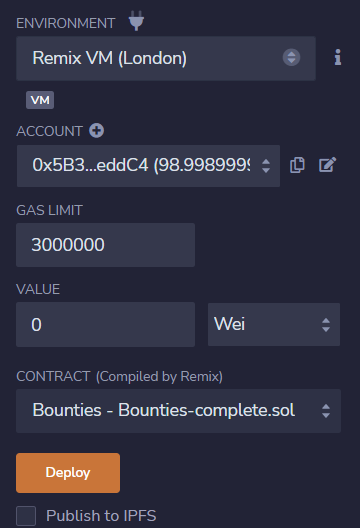


Рисунок 6.2 — Деплой смарт-контракта

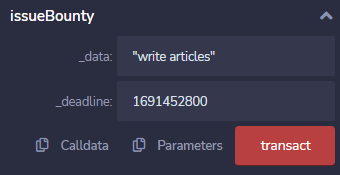


Рисунок 6.3 — Тестирование функции issueBounty

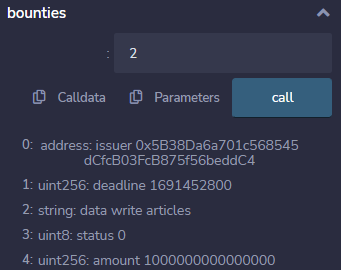


Рисунок 6.4 — Вызов содержимого смарт-контракта по идентификационному номеру

На выходе программа выдала пользователю содержимое смарт-контракта, где указан адрес, срок выполнения работы, описание работы, статус выполнения работы и сумма вознаграждения за выполнение контракта (указана в wei единицах, 1 ETH = 1018 wei).

# Практическая работа №7. Создание реактивного приложения реального времени с использованием Reactor

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**

Был написан реактивный rest api с использованием Spring WebFlux. Ниже приведены листинги модели данных в бд MongoDB (листинг 7.1), контроллер приложения, где описаны следующие методы: получение всех записей, получение записи по id, добавление записи, обновление записи, удаление записи (листинг 7.2). Также приведен код файла запуска и настройки приложения (листинг 7.3).

Листинг 7.1 – User.java

@Document

public class User {

@Id

private String id;

private String name;

private int age;

private double salary;

public User(){

}

public User(String name, int age, double salary){

this.name = name;

this.age = age;

this.salary = salary;

}

public String getId() {

return id;

}

public int getAge() {

return age;

}

public String getName() {

return name;

}

public double getSalary() {

return salary;

}

public void setAge(int age) {

this.age = age;

}

public void setSalary(double salary) {

this.salary = salary;

}

public void setName(String name) {

this.name = name;

}

}

Листинг 7.2 – UserController.java

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired

private UserRepository userRepository;

@PostMapping

public Mono<ResponseEntity<User>> create(@RequestBody User user){

return userRepository.save(user)

.map(savedUser -> ResponseEntity.*ok*(savedUser));

}

@GetMapping

public Flux<User> listUsers(){

return userRepository.findAll();

}

@GetMapping("/{userId}")

public Mono<ResponseEntity<User>> getUserById(@PathVariable String userId){

return userRepository.findById(userId)

.map(user -> ResponseEntity.*ok*(user))

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

}

@PutMapping("/{userId}")

public Mono<ResponseEntity<User>> getUserById(@PathVariable String userId, @RequestBody User user){

return userRepository.findById(userId).flatMap(dbUser -> {

dbUser.setName(user.getName());

dbUser.setAge(user.getAge());

dbUser.setSalary(user.getSalary());

return userRepository.save(dbUser);

})

.map(updatedUser -> ResponseEntity.*ok*(updatedUser))

.defaultIfEmpty(ResponseEntity.*badRequest*().build());

}

@DeleteMapping("/{userId}")

public Mono<ResponseEntity<Void>> deleteUserById(@PathVariable String userId){

return userRepository.findById(userId)

.flatMap(existingUser ->

userRepository.delete(existingUser)

Продолжение листинга 7.2

.then(Mono.*just*(ResponseEntity.*ok*().<Void>build()))

)

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

}

}

Листинг 7.3 - ReactiveApiDemoApplication.java

@SpringBootApplication

public class ReactiveApiDemoApplication {

public static void main(String[] args) {

SpringApplication.*run*(ReactiveApiDemoApplication.class, args);

}

@Bean

CommandLineRunner run(UserRepository userRepository) {

return args -> {

userRepository.deleteAll()

.thenMany(Flux.*just*(

new User("Harrier Du Bois", 44, 1),

new User("Kim Kitsuragi", 43, 3200),

new User("Insulindian Phasmid", 3700, 9000)

)

.flatMap(userRepository::save))

.thenMany(userRepository.findAll())

.subscribe(System.*out*::println);

};

}

}

По завершении разработки было произведено тестирование api в postman (рис. 7.1-7.6).

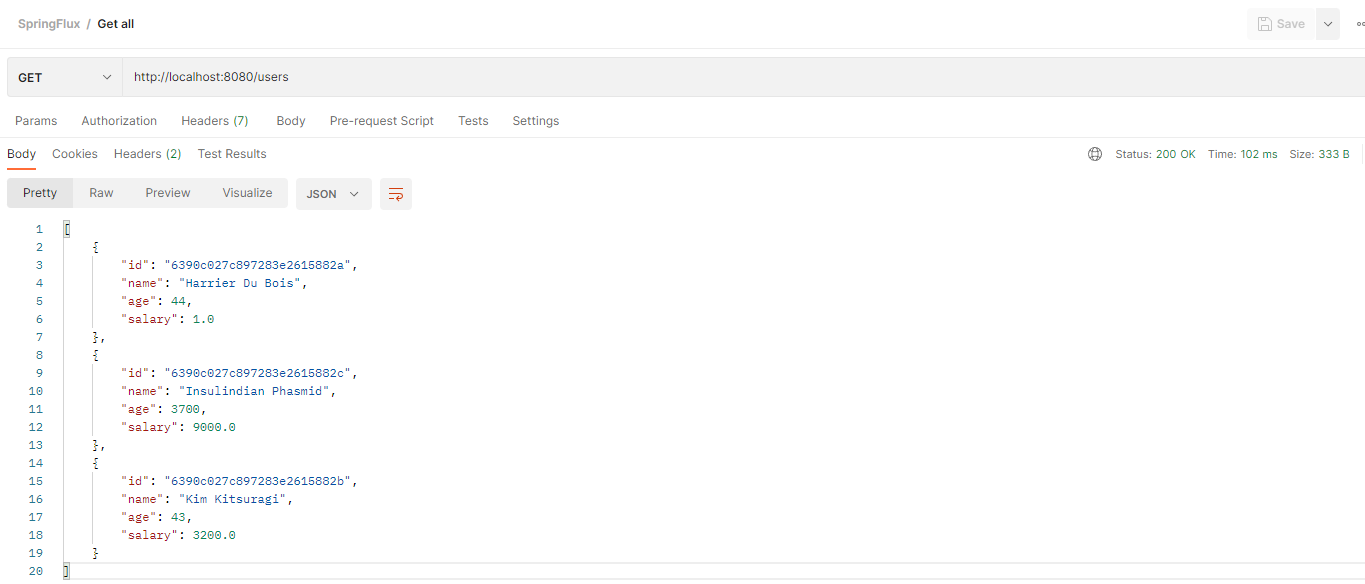


Рисунок 7.1 – Получение всех записей

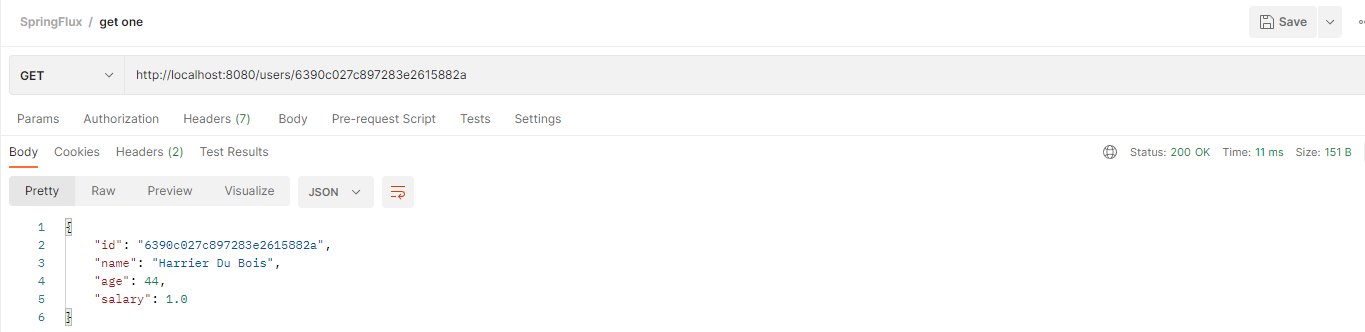


Рисунок 7.2 – Получение одной записи

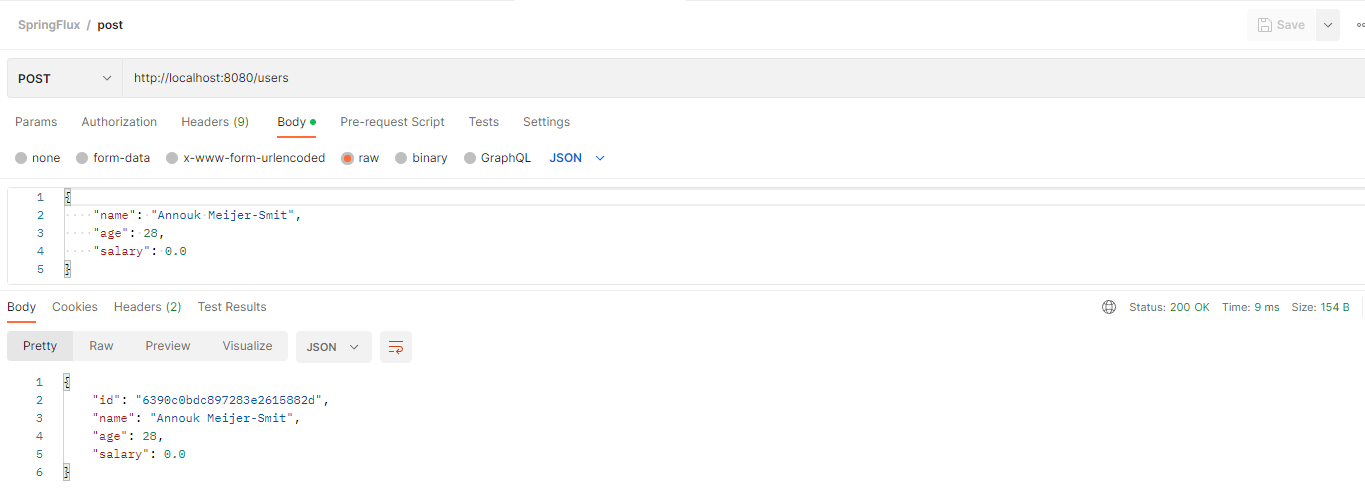


Рисунок 7.3 – Создание новой записи

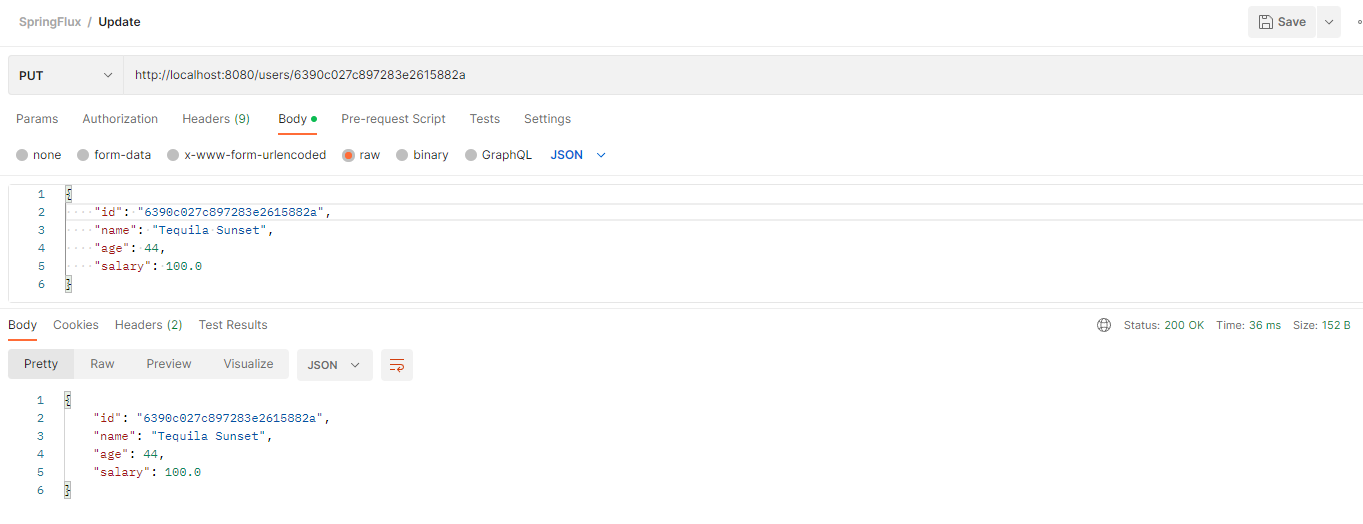


Рисунок 7.4 – Обновление записи

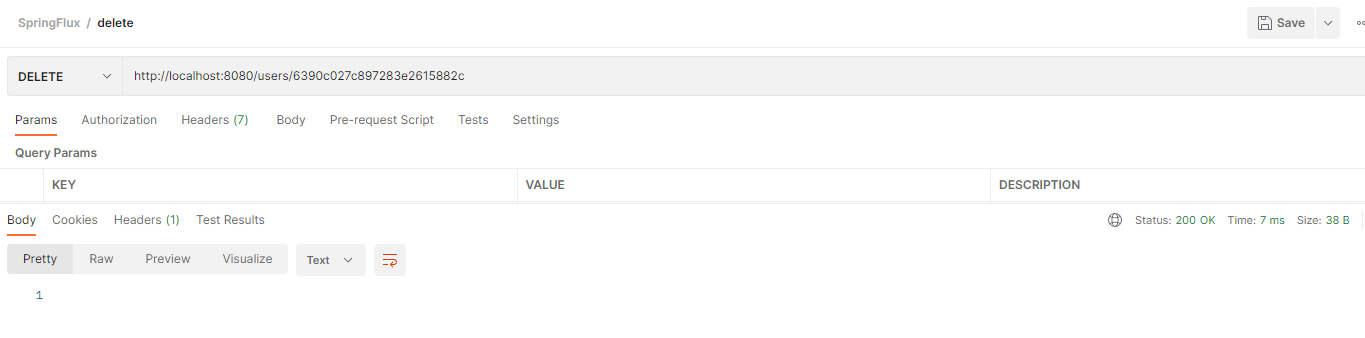


Рисунок 7.5 – Удаление записи

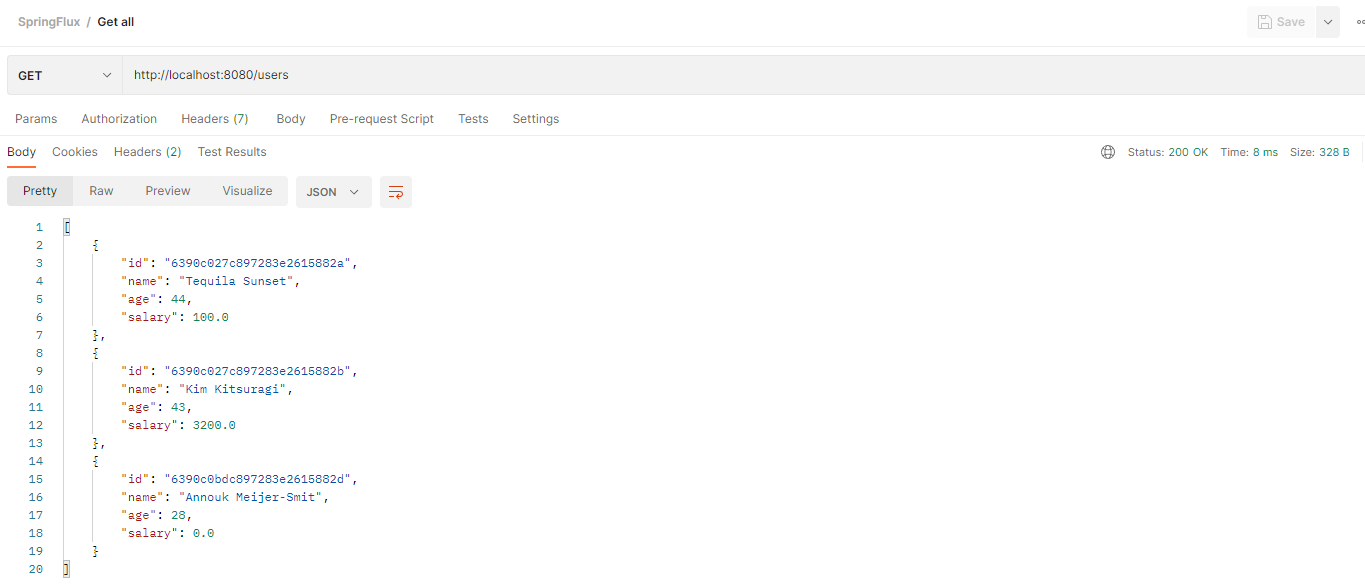


Рисунок 7.6 – Изменения после проведения тестов

# **Вывод**

в ходе выполнения практических работ по дисциплине «Разработка клиент-серверных приложений» были получены полезные теоретические, а главное практические навыки создания приложений, использования многопоточности разными методами (наследованием классов Thread и ForkJoin), создания и обработки файлов, создания и обработки файлов, использования пакета java.nio, а также реализаций копирования методами FileInputStream/FileOutputStream, FileChannel, Apache Commons IO, Files class. В третьей практической работе были использованы классы Observer и Observable из библиотеки RxJava. В четвертой практической работе была разработана клиент-серверная система с использованием протокола RSocket, продемонстрированы 4 типа взаимодействия в рамках протокола (Request-Response, Request-Stream, Channel, Fire-and-Forget). В ходе выполнения пятой практической работы были изучены основные методы работы с Spring/Spring Boot, основными методами протокола и ознакомление с его философией. Освоено взаимодействие протокола с базой данных. В результате выполнения шестой практической работы были получены навыки работы с блокчейн технологиями и языком Solidity, а также использованием среды разработки Remix IDE с применением встроенных инструментов для тестирование реализованных смарт-контрактов. А в результате выполнения седьмой практической работы был реализован сервис, предоставляющий реактивный REST API с использованием Spring WebFlux, удовлетворяющий всем поставленным в задании требованиям.