МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №4

по дисциплине

«Технологии виртуализации клиент-серверных приложений»

Выполнил студент группы ИКБО-01-22 Прокопчук Р.О.

Принял преподаватель кафедры ИиППО Волков М.Ю.

Практические работы выполнены «__»_____2025 г.

«Зачтено» «__»_____2025 г.

Москва
2025

Теоретическое введение

Docker Compose — это инструментальное средство, входящее в состав Docker. Оно предназначено для решения задач, связанных с развёртыванием проектов. Docker Compose позволяет управлять набором контейнеров, каждый из которых представляет собой один сервис проекта. Управление включает в себя сборку, запуск с учетом зависимостей и конфигурацию. Конфигурация Docker Compose описывается в файле docker-compose.yml, лежащем в корне проекта.

Для получения дополнительной информации о файле создания вы можете изучить официальную документацию Docker.

Помимо сервисов на серверах необходимы системы мониторинга, чтобы осуществлять контроль над работоспособностью системы. Для этого используются различные виды систем мониторинга:

— Инфраструктурный мониторинг (Например Zabbix, Nagios)

— Мониторинг ошибок программных платформ (Sentry)

— Мониторинг производительности приложений (Prometheus)

— Мониторинг безопасности систем (Nesus, OpenVas)

— Сбор системных журналов (GrayLog)

Zabbix — это универсальный инструмент мониторинга, способный отслеживать динамику работы серверов и сетевого оборудования, быстро реагировать на внештатные ситуации и предупреждать возможные проблемы с нагрузкой. Система мониторинга Zabbix может собирать статистику в указанной рабочей среде и действовать в определенных случаях заданным образом.

У Zabbix есть 4 основных инструмента, с помощью которых можно мониторить определенную рабочую среду и собирать о ней полный пакет данных для оптимизации работы.

— Сервер — ядро, хранящее в себе все данные системы, включая статистические, оперативные и конфигурацию. Дистанционно управляет

сетевыми сервисами, оповещает администратора о существующих проблемах с оборудованием, находящимся под наблюдением.

— Прокси — сервис, собирающий данные о доступности и производительности устройств, который работает от имени сервера. Все собранные данные сохраняются в буфер и загружаются на сервер. Нужен для распределения нагрузки на сервер. Благодаря этому процессу можно уменьшить нагрузку на процессор и жесткий диск. Для работы прокси Zabbix отдельно нужна база данных.

— Агент — программа (демон), которая активно мониторит и собирает статистику работы локальных ресурсов (накопители, оперативная память, процессор и др.) и приложений.

— Веб-интерфейс — является частью сервера системы и требует для работы веб-сервер. Часто запускается на том же физическом узле, что и Zabbix.

Graylog — это платформа, которая позволяет легко управлять записями структурированных и неструктурированных данных. вместе с отладкой приложений. Он основан на Elasticsearch, MongoDB и Scala.

Он имеет главный сервер, который принимает данные от своих клиентов, установленных на разных серверах, и веб-интерфейс, который отображает данные и позволяет работать с записями, добавленными основным сервером.

Graylog эффективен при работе с необработанными строками (например, с системным журналом) - инструмент анализирует их на нужные нам структурированные данные.

Основное преимущество Graylog заключается в том, что он предоставляет единый идеальный экземпляр сбора журналов для всей системы.

Prometheus — система мониторинга. Основные преимущества — предоставление возможности создания гибких запросов к данным и хранение значений метрик в базе данных временных рядов, возможность автоматизации

при администрировании. Разработана фондом облачных вычислений (Cloud Native Computing Foundation или CNCF).

Для получения метрик с удаленных узлов используется метод pull (сервер сам забирает данные). На узлы для сбора информации устанавливаются экспортеры (exporter) — пакеты, получающие данные для операционной системы или конкретного сервиса. Существует большое количество уже написанных экспортеров для различных приложений. Также метрики могут собираться с помощью механизма push — для этого используется компонент pushgateway, который должен быть установлен дополнительно.

Довольно часто Prometheus настраивают в связке с Grafana, которая позволяет визуализировать показания наших метрик. В Grafana для этого есть уже настроенный источник, таким образом, настройка выполняется из коробки.

Grafana — универсальная обертка для работы с аналитическими данными, которые хранятся в разных источниках. Она сама ничего не хранит и не собирает, а является лишь универсальным клиентом для систем хранения метрик.

Постановка задачи

Вам необходимо создать Spring Boot сервис и произвести мониторинг его работы. В работе должны быть отражены все пункты создания dockercompose файла, проверена работоспособность сервиса и систем мониторинга. Произведена нагрузка на систему с целью проверки работоспособности. Все пункты должны быть отражены в отчете в формате снимков экрана.

Требования к системе:

1. Наличие сервера с CRUD набором для взаимодействия с базой данных и выгрузкой данных с GrayLog по эндпоинту
    a. CRUD набор должен содержать эндпоинты для добавления, обновления, чтения и удаления записей
    b. Количество используемых моделей данных должно быть не менее 3, а также содержать как минимум 1 связь 1-N, 1 связь NN.
    c. При обращении к эндпоинту логов он должен выгружать данные в формате csv из GrayLog
2. PostgreSQL в качестве СУБД с которой будет производиться взаимодействие/сниматься метрики и логи
3. Zabbix для мониторинга базовой работоспособности сервера
4. Prometheus для сбора данных с PostgreSQL, Grafana для вывода этих данных
5. GrayLog для сбора данных с PostgreSQL
6. Adminer для управления базой данных PostgreSQL.

Ход работы

Было разработано приложение, которое хранит информацию о студентах, преподавателях и курсах. Его структура представлена на рисунке 1.

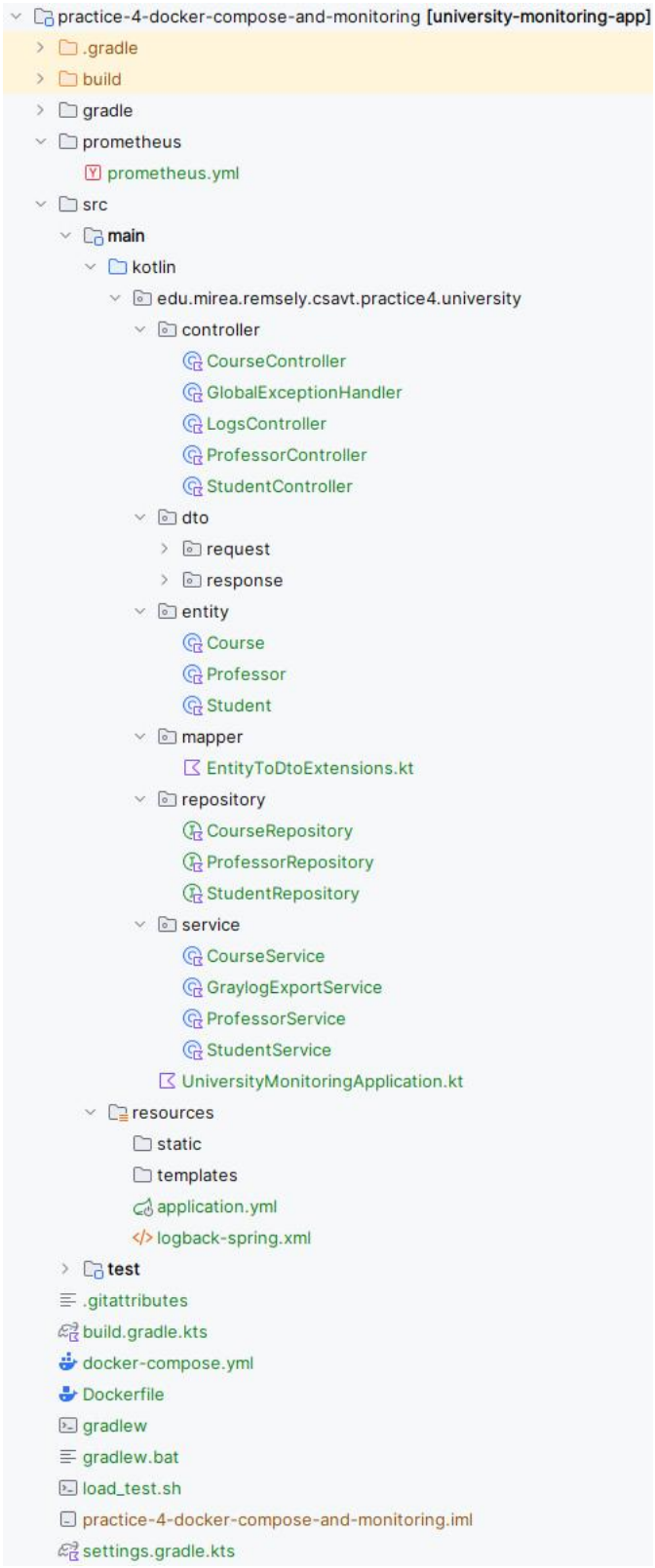

Рисунок 1 – Структура проекта

Файл с зависимостями приложения представлен на рисунке 2.

```
import org.springframework.boot.gradle.plugin.SpringBootPlugin

plugins {
    kotlin("jvm") version "1.9.25"
    kotlin("plugin.spring") version "1.9.25"
    id("org.springframework.boot") version "3.5.6"
    kotlin("plugin.jpa") version "1.9.25"
}

group = "edu.mirea.remsely.csavt.practice4"
version = "0.0.1-SNAPSHOT"
description = "university-system-monitoring"

java {...}

repositories {
    mavenCentral()
}

dependencies { Add Starters...
    implementation(platform( notation = SpringBootPlugin.BOM_COORDINATES))

    implementation("org.springframework.boot:spring-boot-starter-web")
    implementation("org.springframework.boot:spring-boot-starter-webflux")
    implementation("org.springframework.boot:spring-boot-starter-validation")
    implementation("org.springframework.boot:spring-boot-starter-data-jpa")
    implementation("org.springframework.boot:spring-boot-starter-actuator")

    implementation("com.fasterxml.jackson.module:jackson-module-kotlin")
    implementation("io.projectreactor.kotlin:reactor-kotlin-extensions")
    implementation("org.jetbrains.kotlin:kotlin-reflect")
    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-reactor")

    runtimeOnly("org.postgresql:postgresql")

    runtimeOnly("io.micrometer:micrometer-registry-prometheus")
    implementation("de.siegmar:logback-gelf:6.1.2")

    testImplementation("org.springframework.boot:spring-boot-starter-test")
}

kotlin {...}

allOpen {...}

tasks.withType<Test> {...}
```

Рисунок 2 – Класс сущности Course

Код основных сущностей приложения представлен на русунках 3-5.

```kotlin
@Entity  15 Usages  new *
@Table(name = "courses")
open class Course(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id: Long? = null,

    @Column(nullable = false, unique = true)
    var name: String,

    @Column(length = 500)
    var description: String? = null,

    @Column(nullable = false)
    var credits: Int,

    @OneToMany(mappedBy = "course", cascade = [CascadeType.ALL], orphanRemoval = true, fetch = FetchType.LAZY)
    val students: MutableList<Student> = mutableListOf(),

    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable(
        name = "course_professors",
        joinColumns = [JoinColumn(name = "course_id")],
        inverseJoinColumns = [JoinColumn(name = "professor_id")]
    )
    val professors: MutableSet<Professor> = mutableSetOf()
) {
    open fun addProfessor(professor: Professor) {  2 Usages  new *
        professors.add(professor)
        professor.courses.add(this)
    }

    override fun equals(other: Any?): Boolean {  new *
        if (this === other) return true
        if (other !is Course) return false
        return id != null && id == other.id
    }

    override fun hashCode(): Int {  new *
        return javaClass.hashCode()
    }

    override fun toString(): String {  new *
        return "Course(id=$id, name='$name', credits=$credits)"
    }
}
```

Рисунок 3 – Класс сущности Course

```kotlin
@Entity  16 Usages  new *
@Table(name = "professors")
open class Professor(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id: Long? = null,

    @Column(name = "first_name", nullable = false)
    var firstName: String,

    @Column(name = "last_name", nullable = false)
    var lastName: String,

    @Column(unique = true, nullable = false)
    var email: String,

    @Column(nullable = false)
    var department: String,

    @Column(name = "academic_title")
    var academicTitle: String? = null,

    @ManyToMany(mappedBy = "professors", fetch = FetchType.LAZY)
    val courses: MutableSet<Course> = mutableSetOf()
) {
    open val fullName get() = "$firstName $lastName"  new *

    override fun equals(other: Any?): Boolean {  new *
        if (this === other) return true
        if (other !is Professor) return false
        return id != null && id == other.id
    }

    override fun hashCode(): Int {  new *
        return javaClass.hashCode()
    }

    override fun toString(): String {  new *
        return "Professor(id=$id, firstName='$firstName', lastName='$lastName', email='$email')"
    }
}
```

Рисунок 4 – Репозиторий сущности Professor

```kotlin
@Entity  15 Usages  new *
@Table(name = "students")
open class Student(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id: Long? = null,

    @Column(name = "first_name", nullable = false)
    var firstName: String,

    @Column(name = "last_name", nullable = false)
    var lastName: String,

    @Column(unique = true, nullable = false)
    var email: String,

    @Column(name = "student_number", unique = true, nullable = false)
    var studentNumber: String,

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "course_id")
    var course: Course? = null
) {
    override fun equals(other: Any?): Boolean {  new *
        if (this === other) return true
        if (other !is Student) return false
        return id != null && id == other.id
    }

    override fun hashCode(): Int {  new *
        return javaClass.hashCode()
    }

    override fun toString(): String {  new *
        return "Student(id=$id, firstName='$firstName', lastName='$lastName', email='$email')"
    }
}
```

Рисунок 5 – Класс сущности Student

Конфигурация Prometheus представлена на рисунке 6.

```yaml
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: 'spring-boot-app'
    metrics_path: '/actuator/prometheus'
    static_configs:
      - targets: ['app:8080']
        labels:
          application: 'university-monitoring-app'

  - job_name: 'postgres-exporter'
    static_configs:
      - targets: ['postgres-exporter:9187']
        labels:
          database: 'monitoring_db'

  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
```

Рисунок 6 – Конфигурация Prometheus

Конфигурация docker-compose представлена на рисунках 7-11.

```yaml
services:
  app:
    build: .
    container_name: monitoring-app
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://postgres:5432/monitoring_db
      SPRING_DATASOURCE_USERNAME: postgres
      SPRING_DATASOURCE_PASSWORD: postgres123
      GRAYLOG_HOST: graylog
      GRAYLOG_PORT: 12201
    ports:
      - "8080:8080"
    depends_on:
      postgres:
        condition: service_healthy
      graylog:
        condition: service_started
    networks:
      - monitoring-network
```

Рисунок 7 – Конфигурация Spring Boot приложения

```yaml
services:
  postgres:
    image: postgres:16
    container_name: monitoring-postgres
    environment:
      POSTGRES_DB: monitoring_db
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres123
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - monitoring-network
    healthcheck:
      test: [ "CMD-SHELL", "pg_isready -U postgres" ]
      interval: 10s
      timeout: 5s
      retries: 5

  postgres-exporter:
    image: prometheuscommunity/postgres-exporter:v0.18.1
    container_name: monitoring-postgres-exporter
    environment:
      DATA_SOURCE_NAME: "postgresql://postgres:postgres123@postgres:5432/monitoring_db?sslmode=disable"
    ports:
      - "9187:9187"
    networks:
      - monitoring-network
    depends_on:
      postgres:
        condition: service_healthy

  adminer:
    image: adminer:5.4.0
    container_name: monitoring-adminer
    ports:
      - "8081:8080"
    networks:
      - monitoring-network
    depends_on:
      - postgres
```

Рисунок 8 – Конфигурация СУБД PostgreSQL и сервисов ее мониторинга

```yaml
prometheus:
  image: prom/prometheus:v3.6.0
  container_name: monitoring-prometheus
  volumes:
    - ./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml
    - prometheus_data:/prometheus
  command:
    - '--config.file=/etc/prometheus/prometheus.yml'
    - '--storage.tsdb.path=/prometheus'
  ports:
    - "9090:9090"
  networks:
    - monitoring-network
  depends_on:
    - app

grafana:
  image: grafana/grafana:12.3.0-18329792253
  container_name: monitoring-grafana
  environment:
    GF_SECURITY_ADMIN_USER: admin
    GF_SECURITY_ADMIN_PASSWORD: admin
  ports:
    - "3000:3000"
  volumes:
    - grafana_data:/var/lib/grafana
  networks:
    - monitoring-network
  depends_on:
    - prometheus
```

Рисунок 9 – Конфигурация Prometheus и Grafana

```yaml
services:
  graylog:
    image: graylog/graylog:5.2
    container_name: monitoring-graylog
    environment:
      GRAYLOG_PASSWORD_SECRET: "somepasswordpepper12345678901234567890123456789012345678901234567890
      GRAYLOG_ROOT_PASSWORD_SHA2: "8c6976e5b5410415bde908bd4dee15dfb167a9c873fc4bb8a81f6f2ab448a918"
      GRAYLOG_HTTP_EXTERNAL_URI: "http://localhost:9000/"
      GRAYLOG_MONGODB_URI: "mongodb://mongodb:27017/graylog"
      GRAYLOG_ELASTICSEARCH_HOSTS: "http://elasticsearch:9200"
    ports:
      - "9000:9000"
      - "12201:12201/udp"
      - "1514:1514"
    volumes:
      - graylog_data:/usr/share/graylog/data
    networks:
      - monitoring-network
    depends_on:
      - mongodb
      - elasticsearch

  mongodb:
    image: mongo:7
    container_name: monitoring-mongodb
    environment:
      MONGO_INITDB_DATABASE: graylog
    volumes:
      - mongodb_data:/data/db
    networks:
      - monitoring-network

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.17.18
    container_name: monitoring-elasticsearch
    environment:
      - discovery.type=single-node
      - xpack.security.enabled=false
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    volumes:
      - elasticsearch_data:/usr/share/elasticsearch/data
    networks:
      - monitoring-network
```

Рисунок 10 – Конфигурация GrayLog и его зависимостей

```yaml
services:
  zabbix-server:
    image: zabbix/zabbix-server-pgsql:latest
    container_name: monitoring-zabbix-server
    environment:
      DB_SERVER_HOST: zabbix-postgres
      POSTGRES_DB: zabbix
      POSTGRES_USER: zabbix
      POSTGRES_PASSWORD: zabbix123
    ports:
      - "10051:10051"
    networks:
      - monitoring-network
    depends_on:
      - zabbix-postgres

  zabbix-postgres:
    image: postgres:16
    container_name: monitoring-zabbix-postgres
    environment:
      POSTGRES_DB: zabbix
      POSTGRES_USER: zabbix
      POSTGRES_PASSWORD: zabbix123
    volumes:
      - zabbix_postgres_data:/var/lib/postgresql/data
    networks:
      - monitoring-network

  zabbix-web:
    image: zabbix/zabbix-web-nginx-pgsql:latest
    container_name: monitoring-zabbix-web
    environment:
      ZBX_SERVER_HOST: zabbix-server
      DB_SERVER_HOST: zabbix-postgres
      POSTGRES_DB: zabbix
      POSTGRES_USER: zabbix
      POSTGRES_PASSWORD: zabbix123
      PHP_TZ: Europe/Moscow
    ports:
      - "8082:8080"
    networks:
      - monitoring-network
    depends_on:
      - zabbix-server

  zabbix-agent:
    image: zabbix/zabbix-agent2:latest
    container_name: monitoring-zabbix-agent
    environment:
      ZBX_HOSTNAME: "monitoring-app"
      ZBX_SERVER_HOST: zabbix-server
      ZBX_SERVER_PORT: 10051
      ZBX_PASSIVE_ALLOW: "true"
      ZBX_ACTIVE_ALLOW: "true"
    ports:
      - "10050:10050"
    networks:
      - monitoring-network
```

Рисунок 11 – Конфигурация Zabbix и связанных контейнеров

Запущенные контейнеры представлены на рисунке 12.

```
[+] Running 22/22
 ✓ app                                                                    Built
 ✓ Network practice-4-docker-compose-and-monitoring_monitoring-network    Created
 ✓ Volume "practice-4-docker-compose-and-monitoring_mongodb_data"         Created
 ✓ Volume "practice-4-docker-compose-and-monitoring_grafana_data"         Created
 ✓ Volume "practice-4-docker-compose-and-monitoring_graylog_data"         Created
 ✓ Volume "practice-4-docker-compose-and-monitoring_elasticsearch_data"   Created
 ✓ Volume "practice-4-docker-compose-and-monitoring_prometheus_data"      Created
 ✓ Volume "practice-4-docker-compose-and-monitoring_zabbix_postgres_data" Created
 ✓ Volume "practice-4-docker-compose-and-monitoring_postgres_data"        Created
 ✓ Container monitoring-mongodb                                           Started
 ✓ Container monitoring-elasticsearch                                     Started
 ✓ Container monitoring-zabbix-postgres                                   Started
 ✓ Container monitoring-postgres                                          Healthy
 ✓ Container monitoring-zabbix-server                                     Started
 ✓ Container monitoring-adminer                                           Started
 ✓ Container monitoring-postgres-exporter                                 Started
 ✓ Container monitoring-graylog                                           Started
 ✓ Container monitoring-zabbix-agent                                      Started
 ✓ Container monitoring-zabbix-web                                        Started
 ✓ Container monitoring-app                                               Started
 ✓ Container monitoring-prometheus                                        Started
 ✓ Container monitoring-grafana                                           Started
```

Рисунок 12 – Запущенные контейнеры

Примеры POST-запросов к сервису представлены на рисунках 13-15.

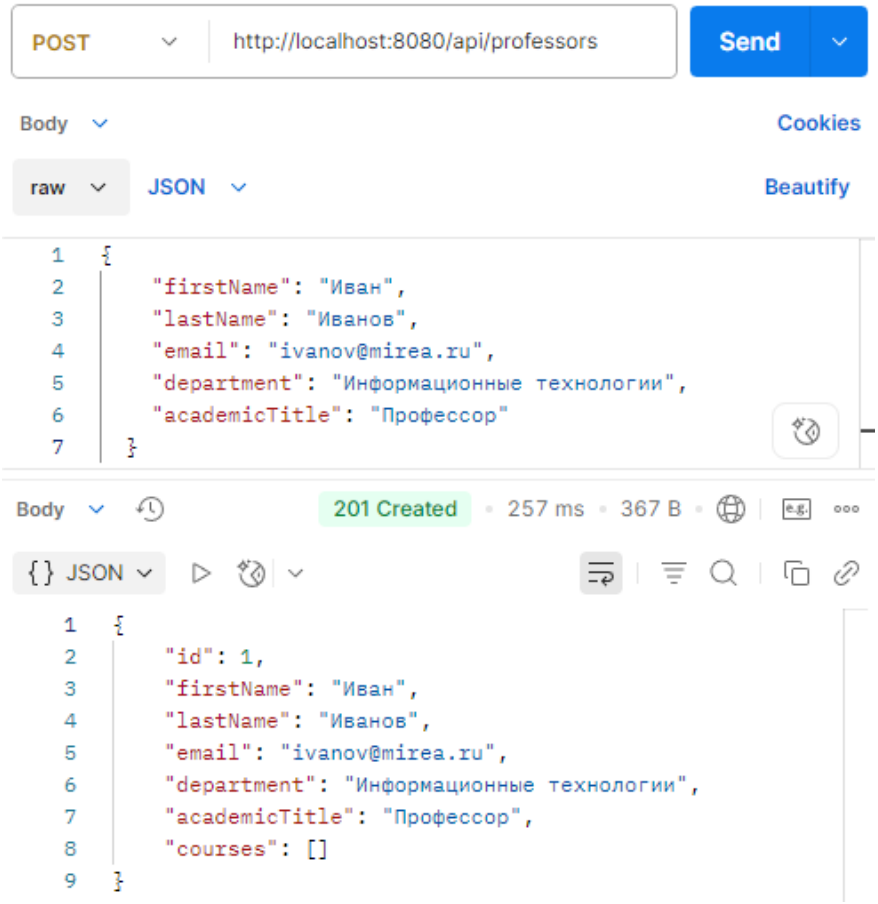

Рисунок 13 – POST-запрос для добавления преподавателя

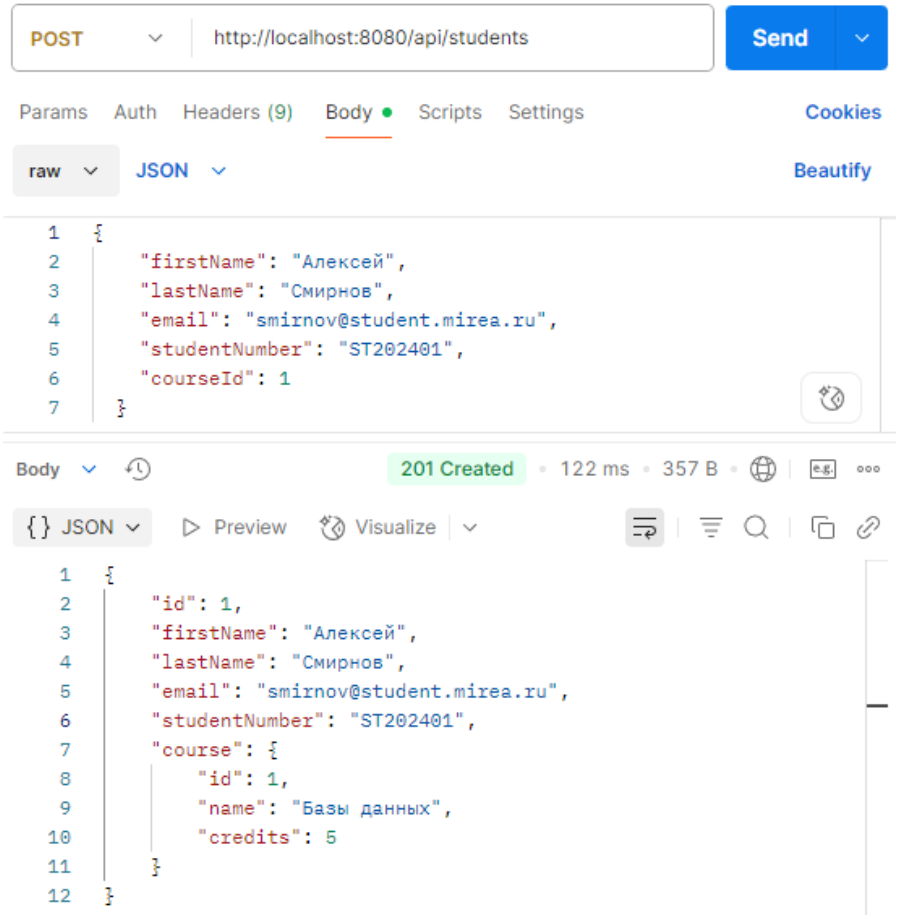

Рисунок 14 – POST-запрос для добавления студента

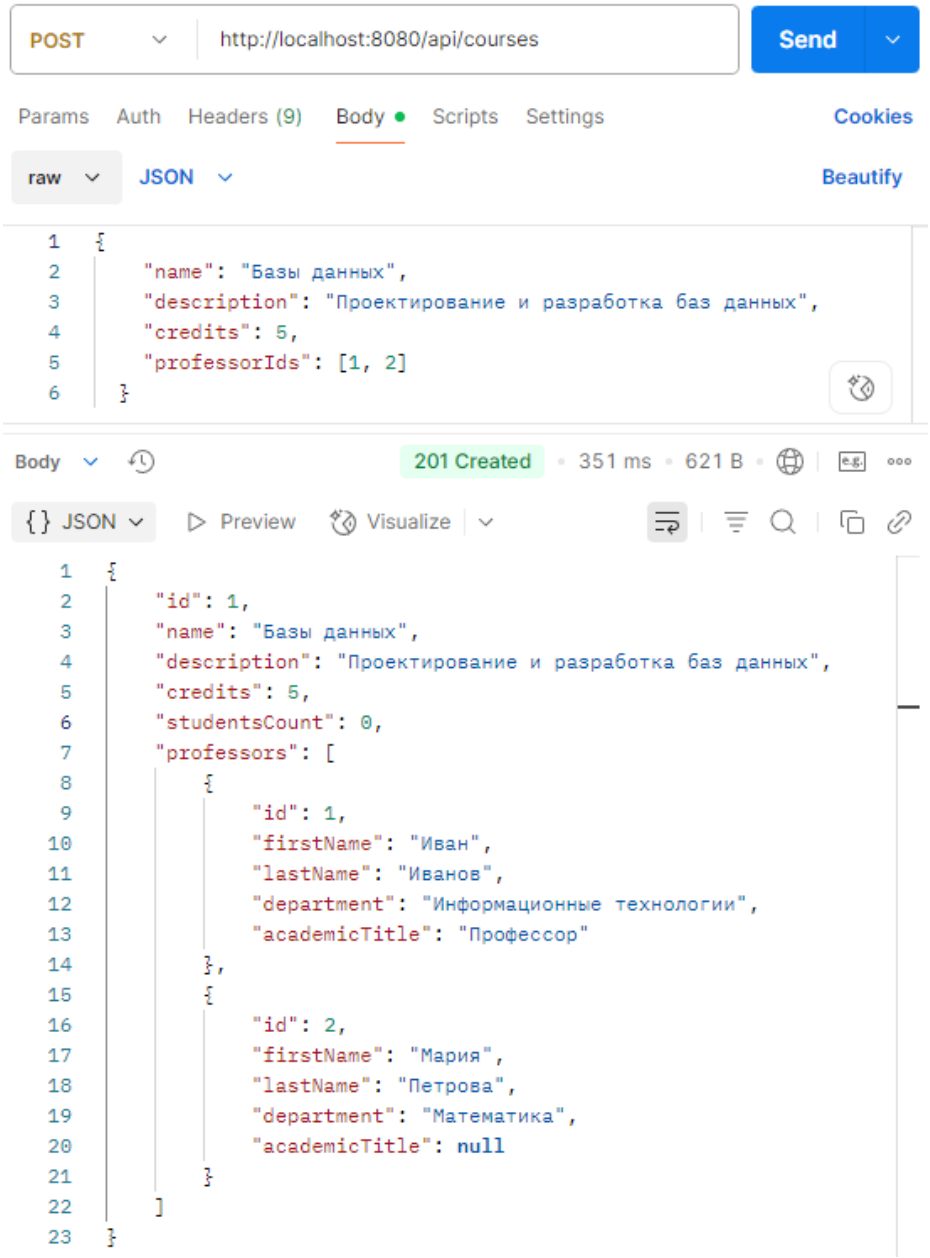

Рисунок 15 – POST-запрос для добавления курса

Примеры GET-запросов к сервису представлены на рисунках 16-17.

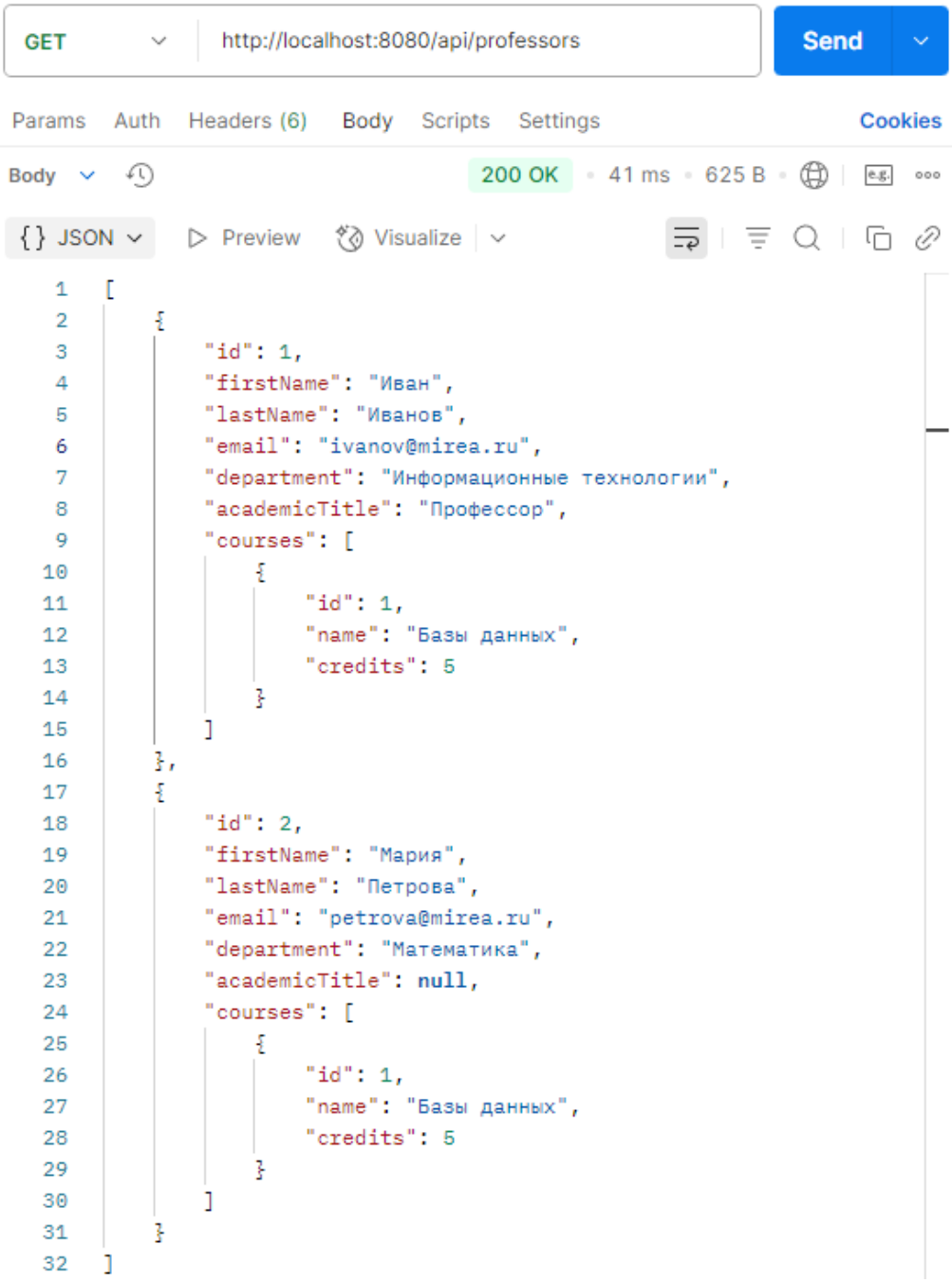

Рисунок 16 – GET-запрос для получения преподавателей

Рисунок 17 – GET-запрос для получения студентов курса

Пример PUT-запроса к сервису представлен на рисунке 18.

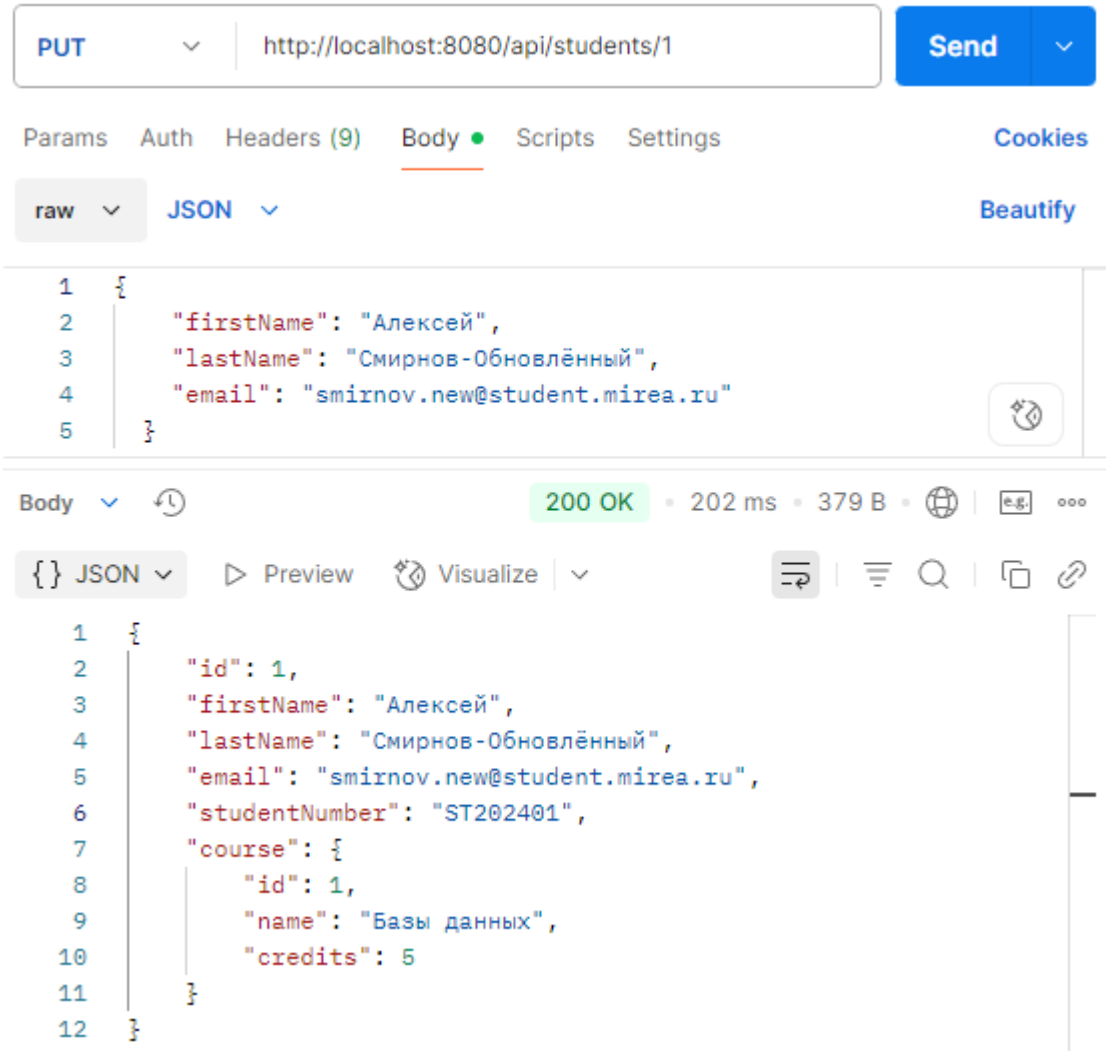

Рисунок 18 – PUT-запрос для обновления студента

Пример DELETE-запроса к сервису представлен на рисунке 19 (GET-запрос для подтверждения удаления представлен на рисунке 20).

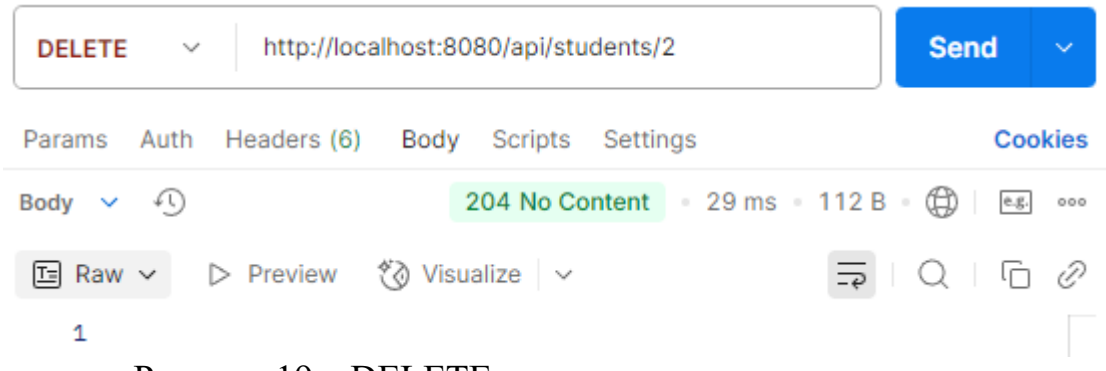

Рисунок 19 – DELETE-запрос для удаления студента

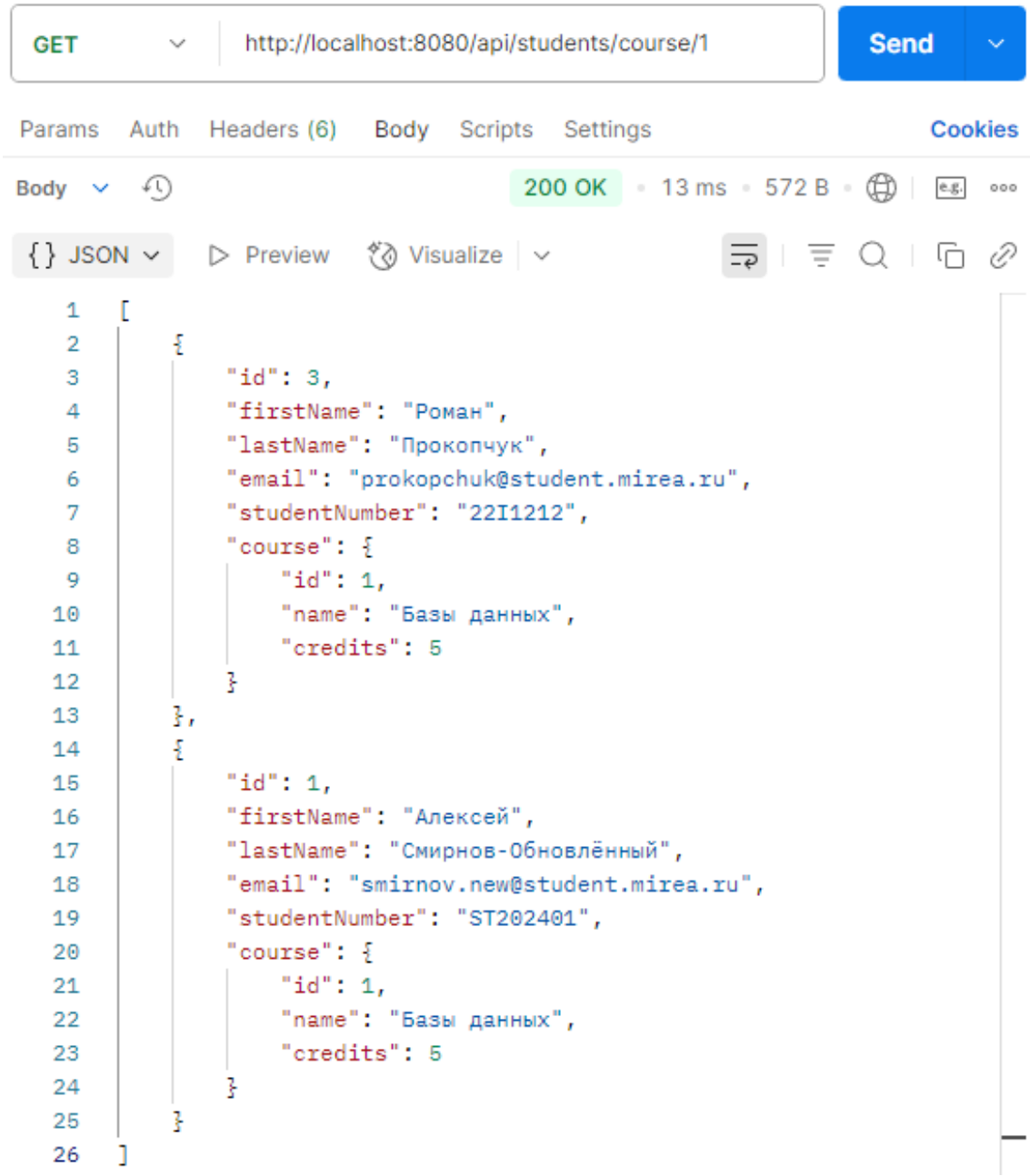

Рисунок 20 – Подтверждающий удаление GET-запрос

Запрос к Actuator представлен на рисунке 21.

Информация о БД в Adminer представлен на рисунке 22.

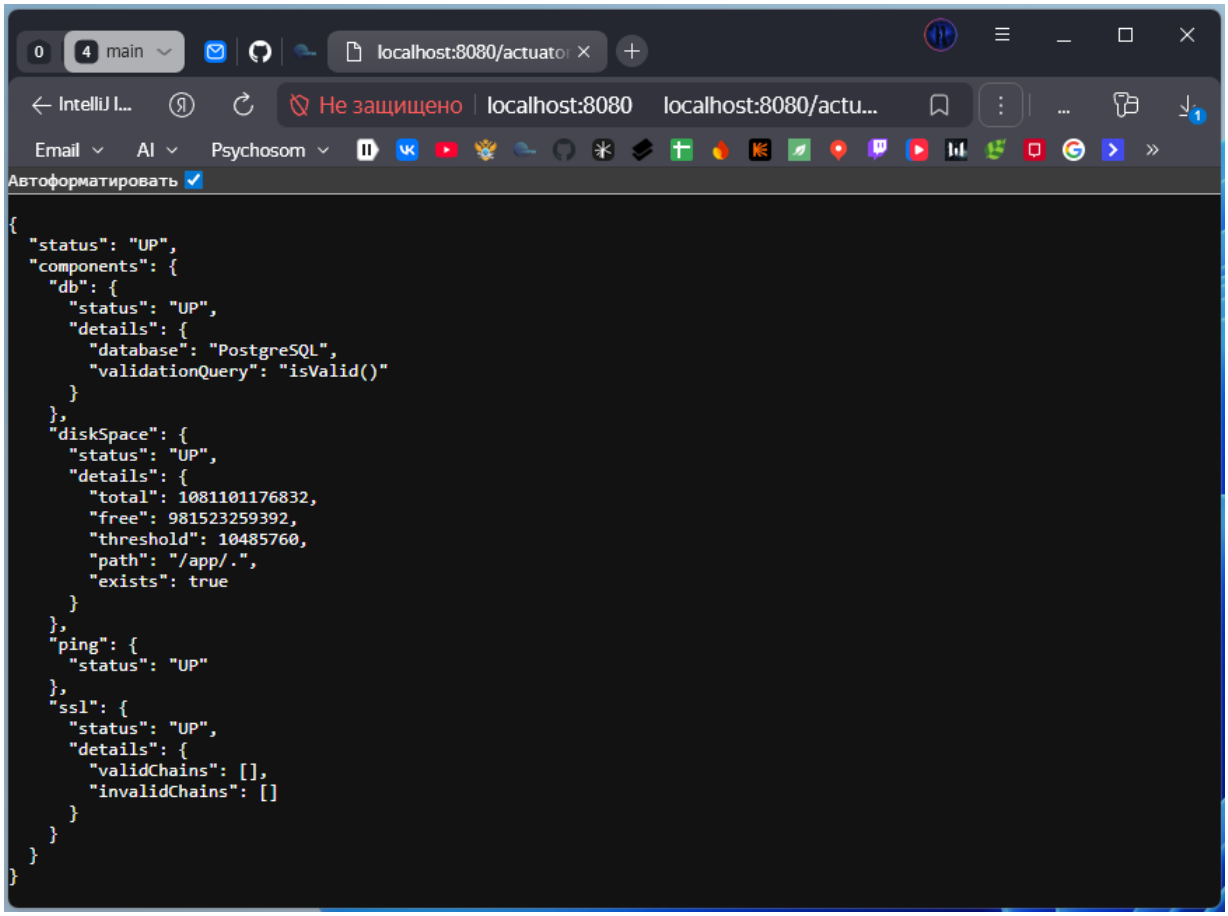

Рисунок 21 – Запрос к Actuator

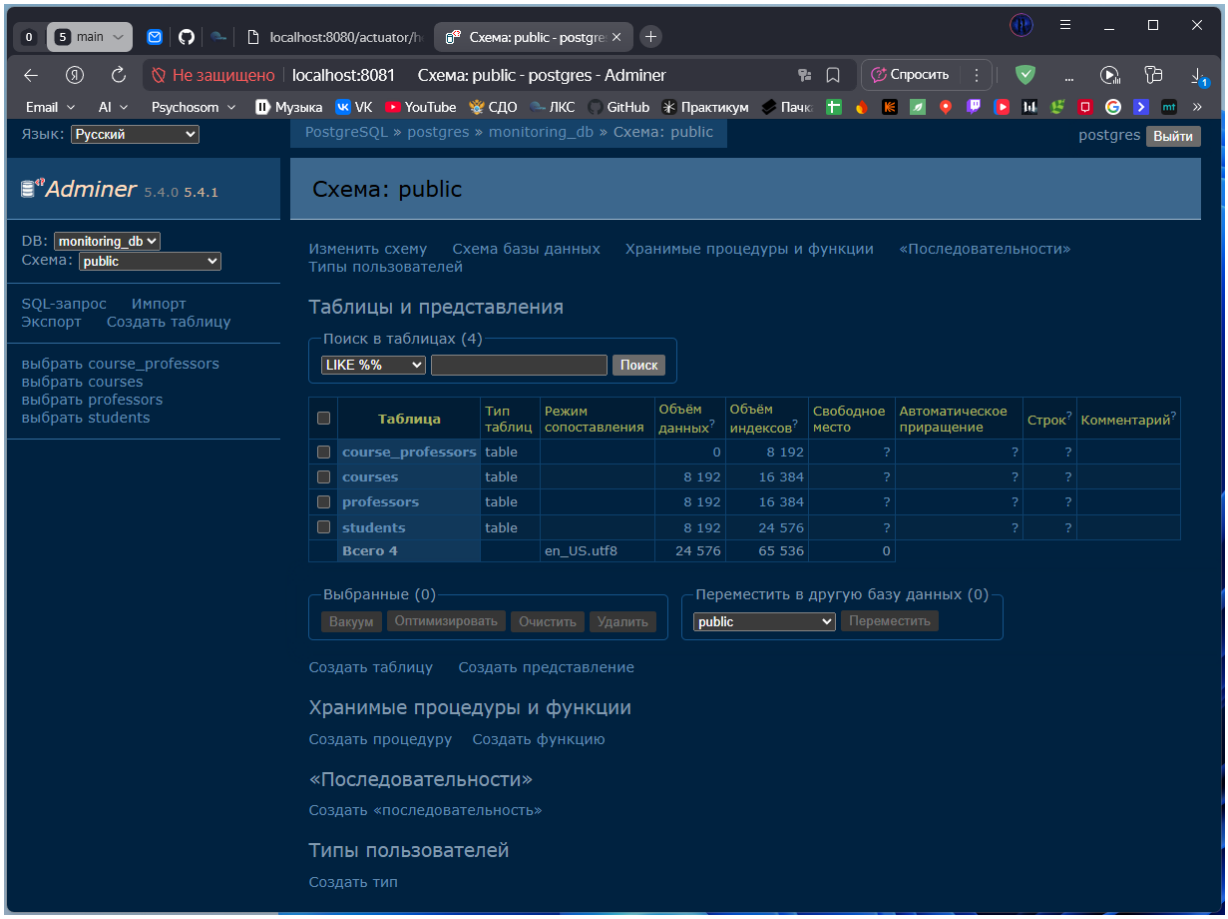

Рисунок 22 – Информация о БД в Adminer

Информация о метриках в Prometheus предтавлена на рисунке 23.

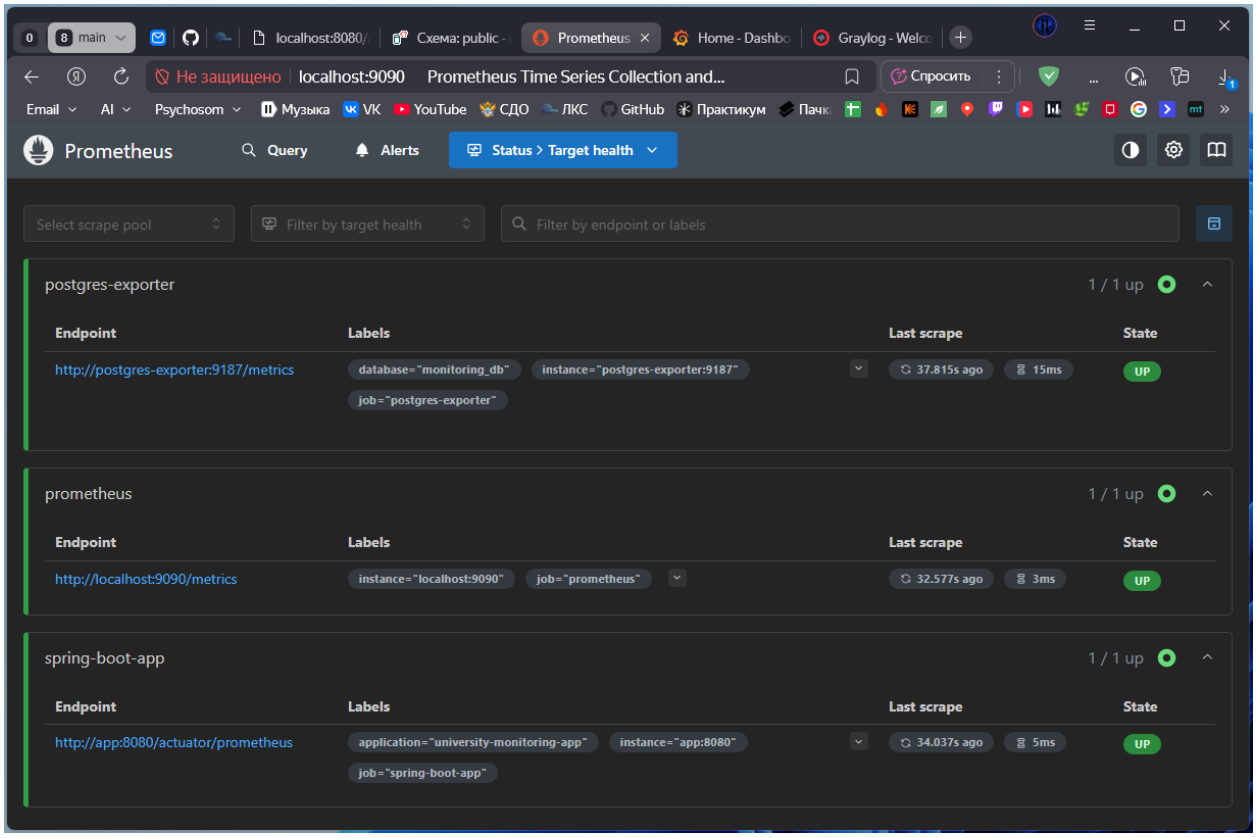

Рисунок 23 – Информация о метриках в Prometheus

Пример потока метрик представлен на рисунке 24.

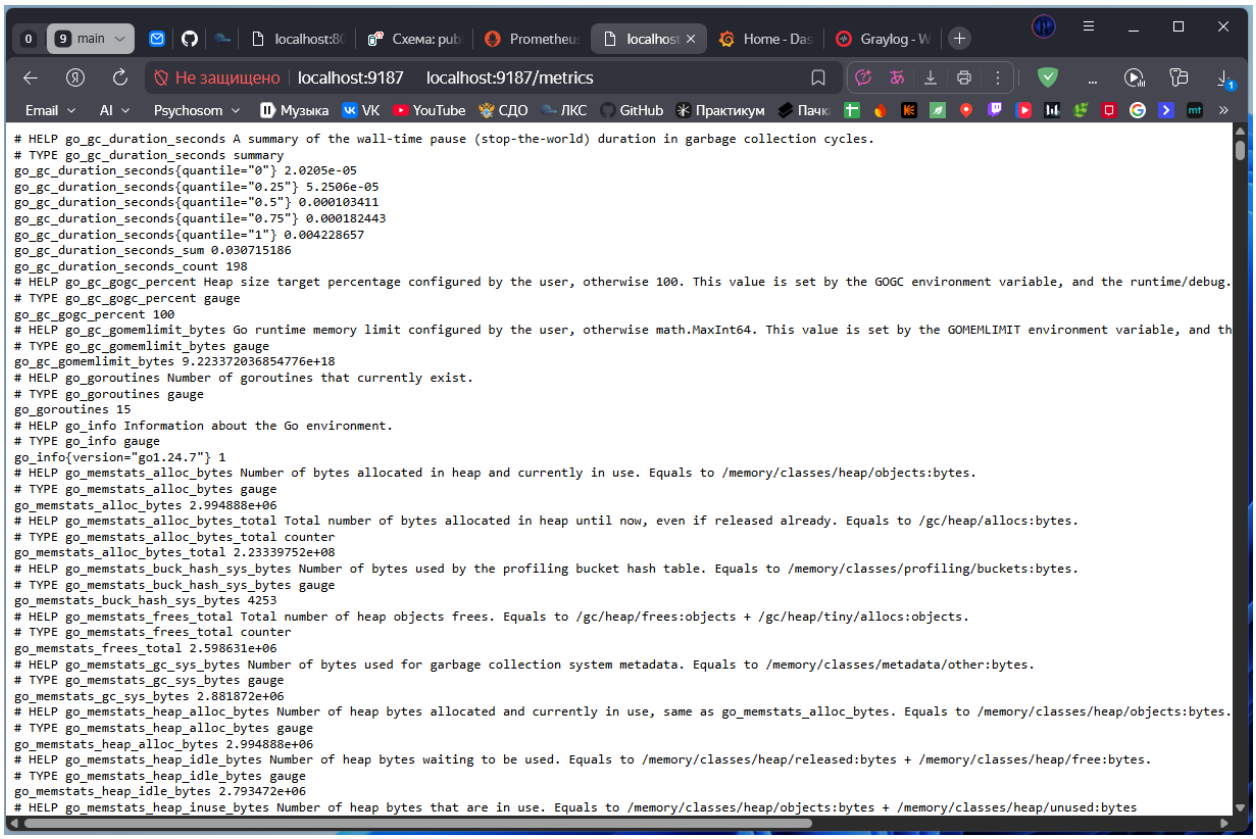

Рисунок 24 – Пример потока метрик

Импортирование дашборда для PostgreSQL представлено на рисунке 25.

Итоговый список дашбордов представлен на рисунке 26.

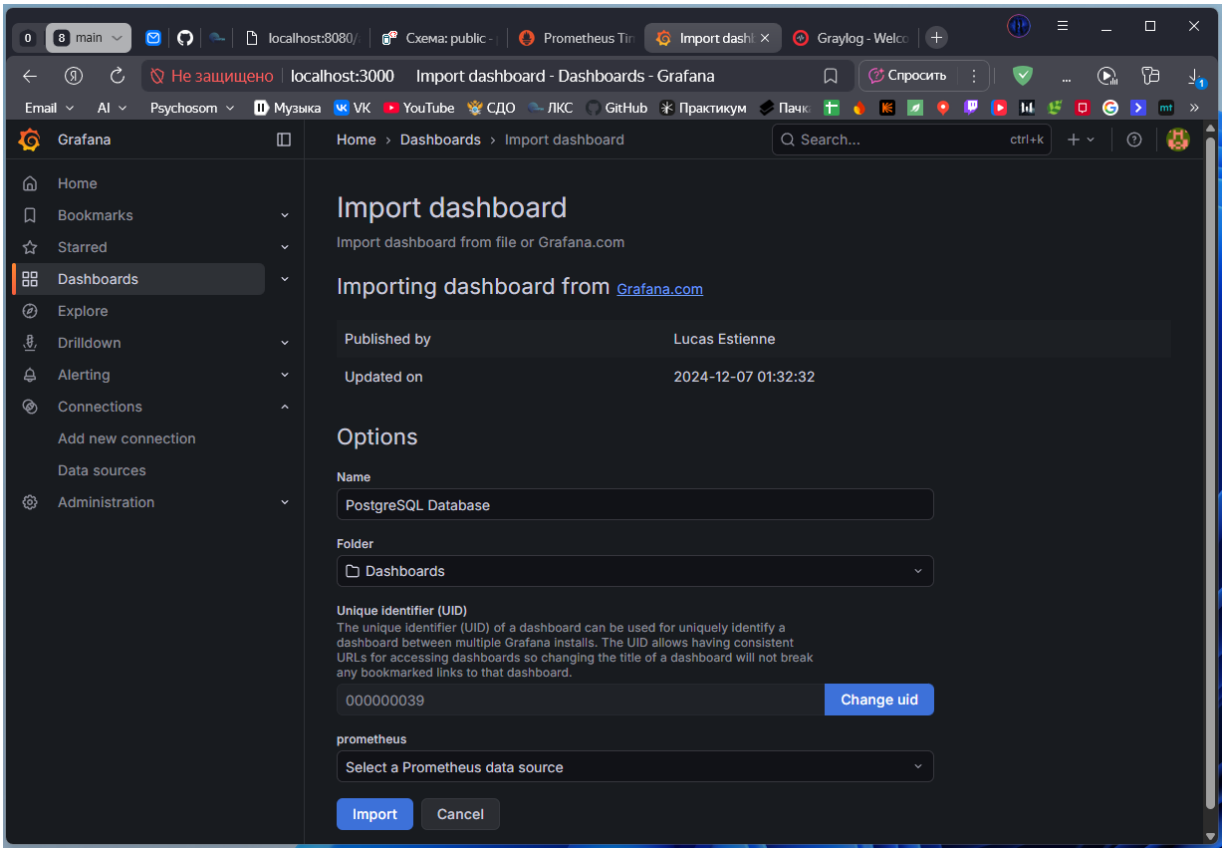

Рисунок 25 – Импорт дашборда PostgreSQL

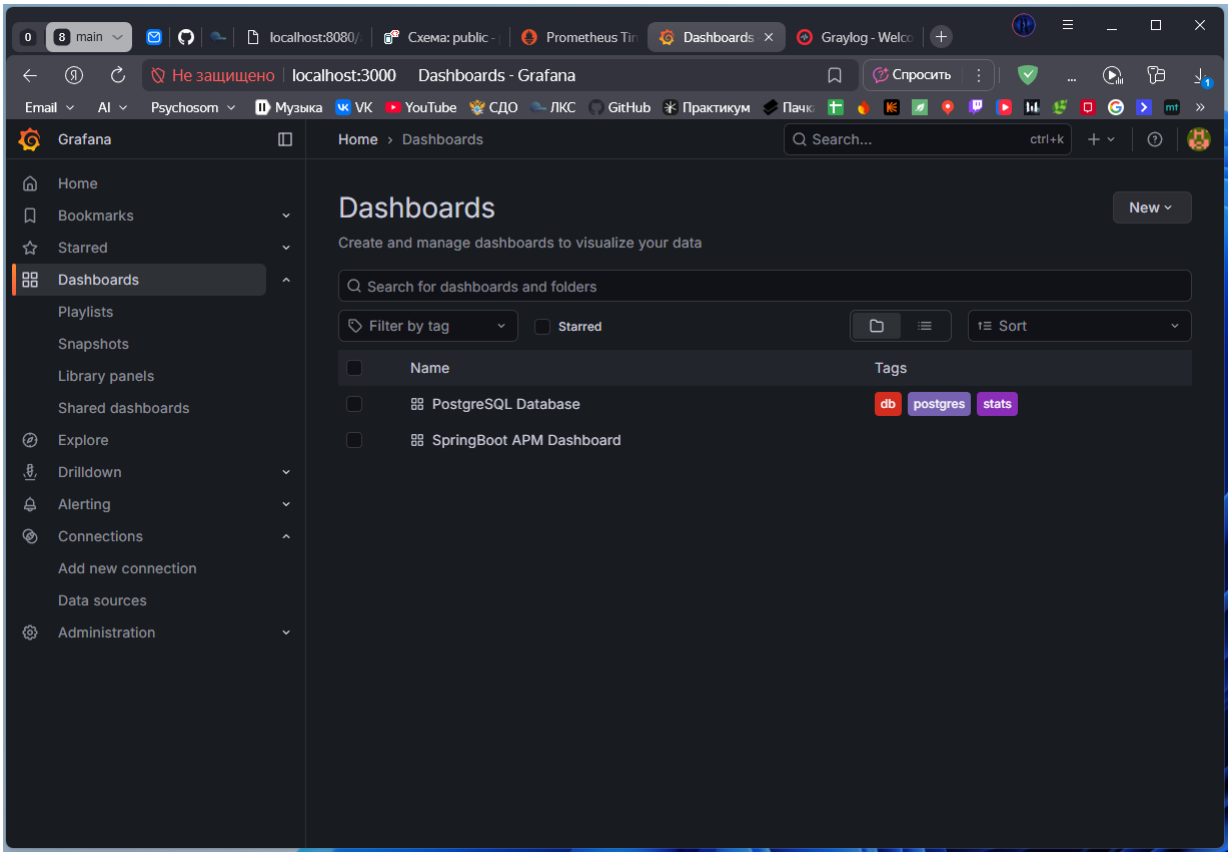

Рисунок 26 – Итоговый список дашбордов

Дашборд с метриками PostgreSQL представлен на рисунке 27, дашборд с метриками Spring Boot приложения – на рисунке 28.

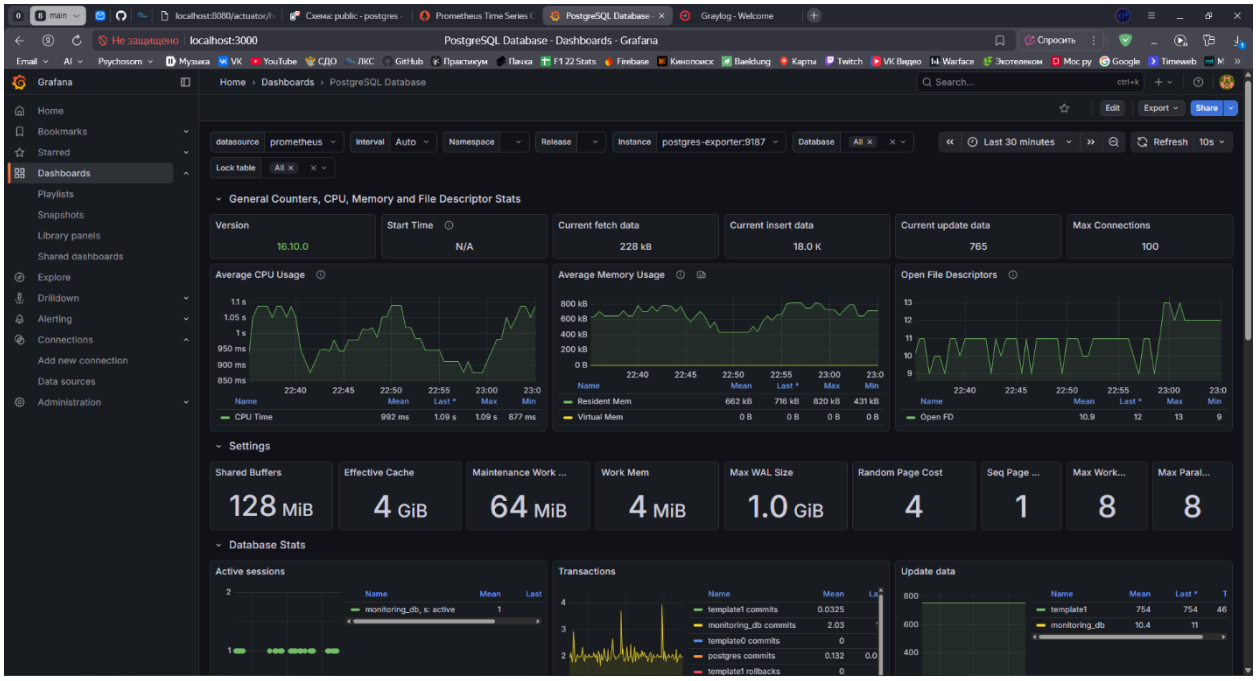

Рисунок 27 – Дашборд с метриками PostgreSQL

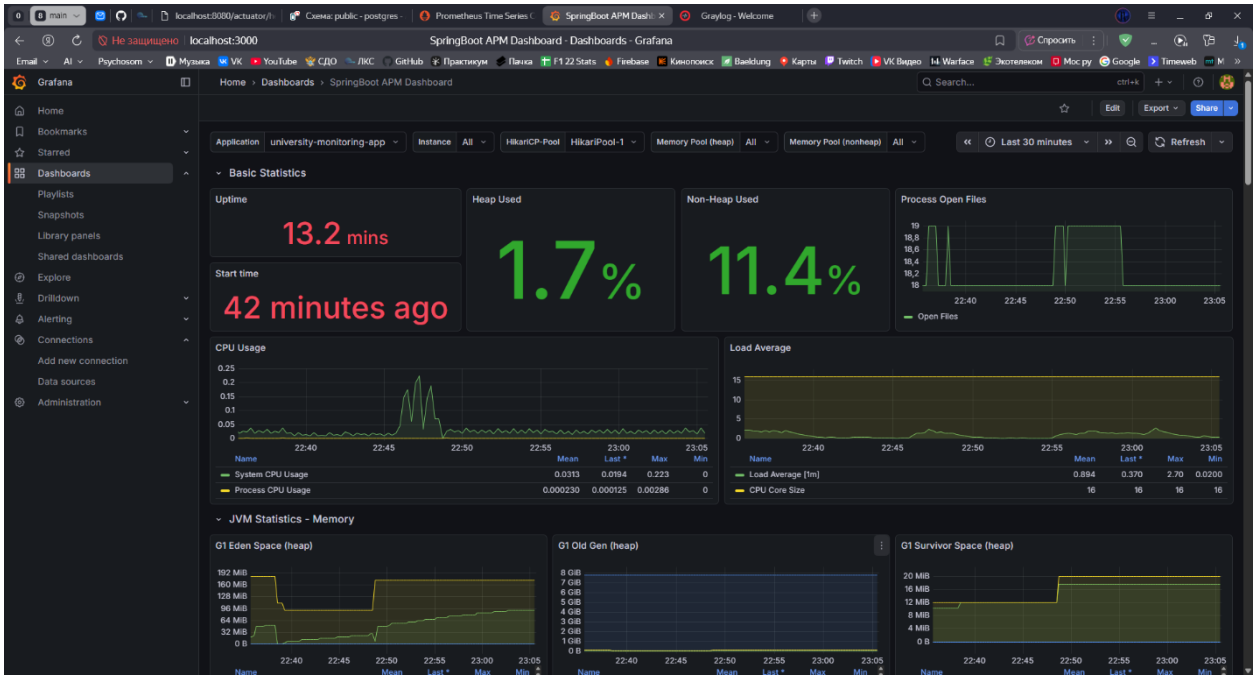

Рисунок 28 – Дашборд с метриками PostgreSQL

Логи в интерфейсе GrayLog представлены на рисунке 29, логи, полученные в формате CSV – на рисунке 30.

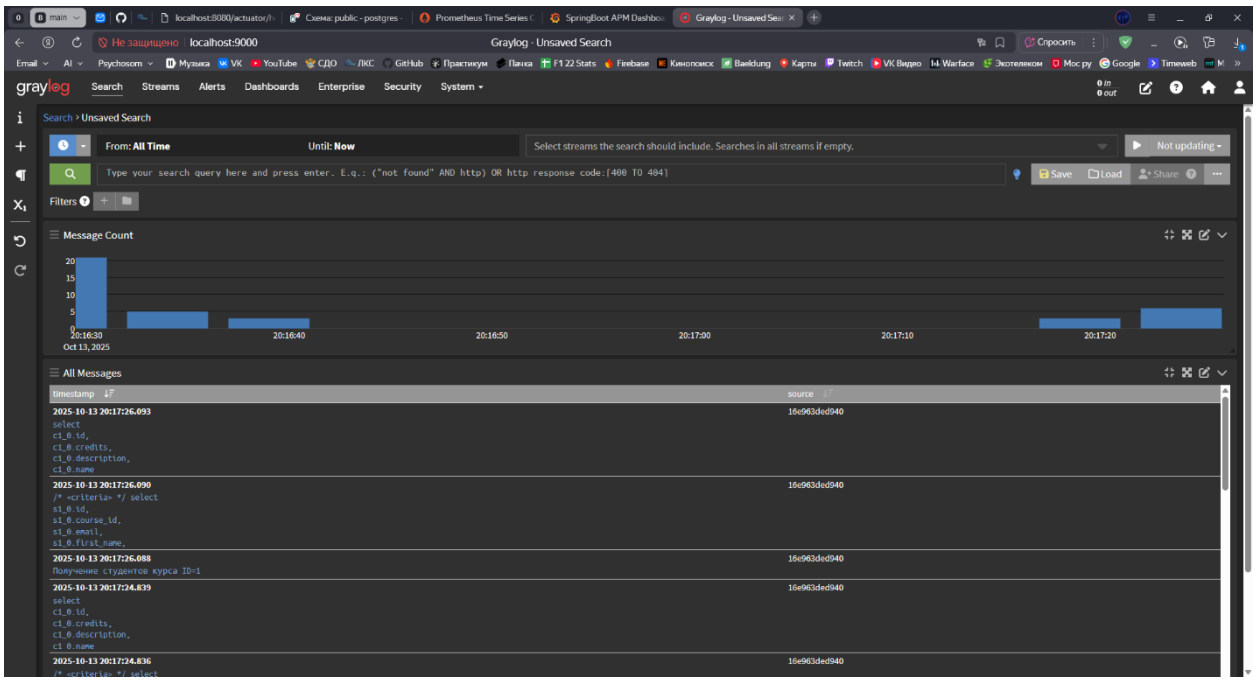

Рисунок 29 – Логи приложения в интерфейсе GrayLog

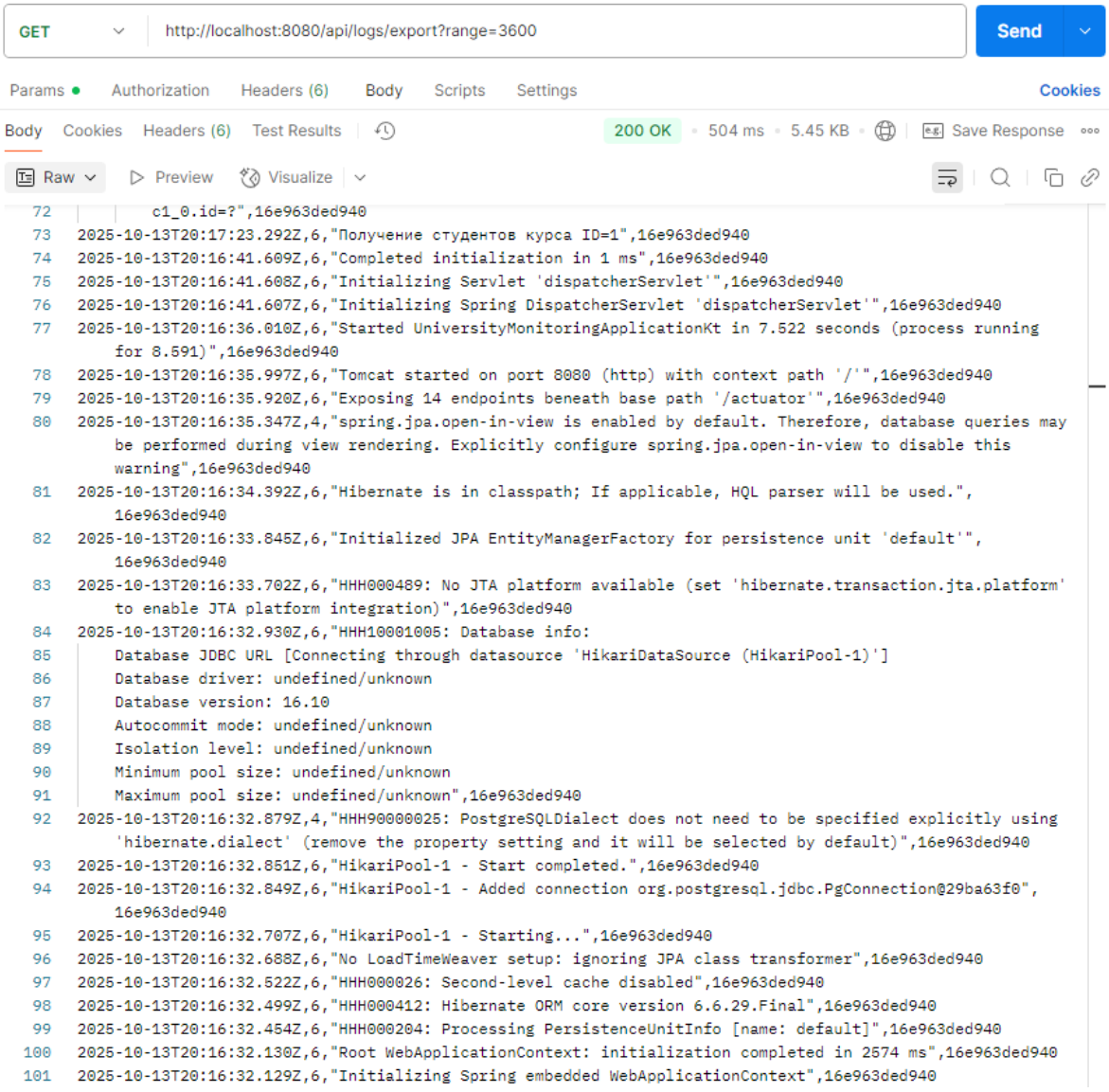

Рисунок 30 – Логи, полученные в формате CSV

Информация о приложении в Zabbix представлена на рисунке 31.

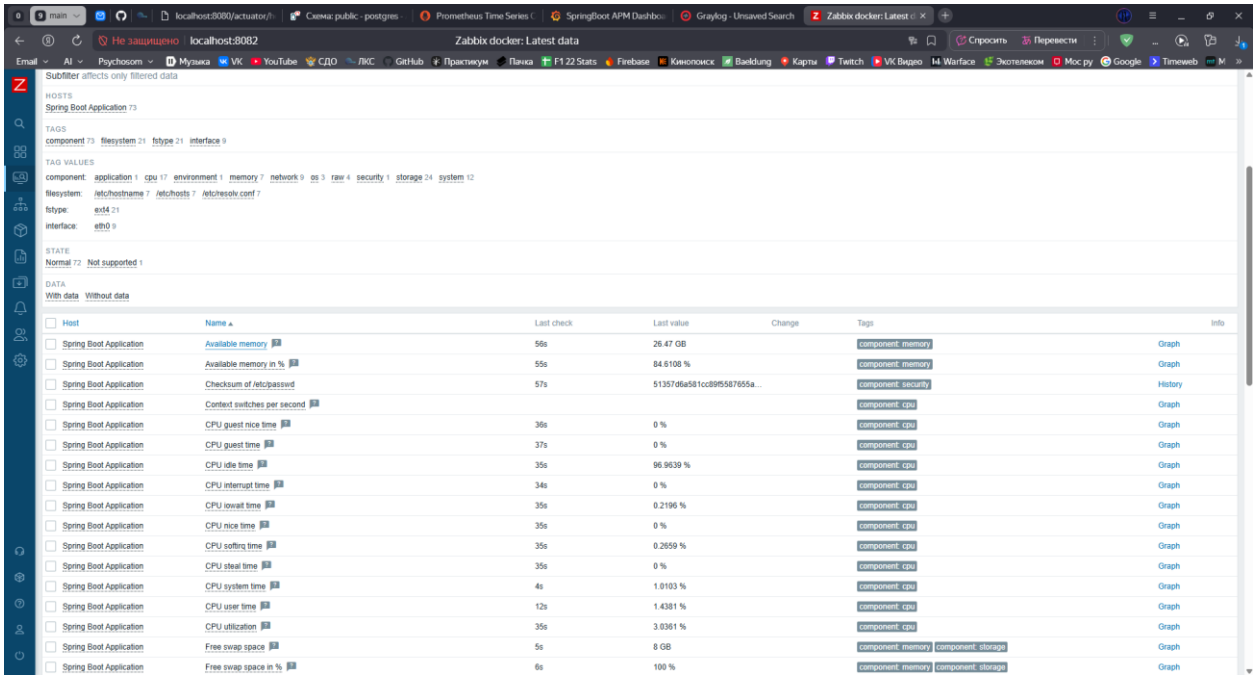

Рисунок 31 – Информация о приложении в интерфейсе Zabbix

Скрипт для подачи нагрузки на приложение представлен на рисунке 32.

load_test.sh

```
#!/bin/bash

echo "Начало нагрузочного теста..."

for i in {1..100}
do
  curl -X POST http://localhost:8080/api/students \
    -H "Content-Type: application/json" \
    -d "{
      \"firstName\": \"Student$i\",
      \"lastName\": \"Test$i\",
      \"email\": \"test$i@student.mirea.ru\",
      \"studentNumber\": \"ST$i\",
      \"courseId\": 1
    }" &

  curl http://localhost:8080/api/students > /dev/null 2>&1 &

  curl http://localhost:8080/api/courses > /dev/null 2>&1 &

  if [ $((i % 10)) -eq 0 ]; then
    echo "Выполнено запросов: $i"
    sleep 1
  fi
done

wait
echo "Нагрузочный тест завершён!"
```

Рисунок 32 – Скрипт для тестирования нагрузки

Запуск скрипта для нагрузочного тестирования представлен на рисунке 33.

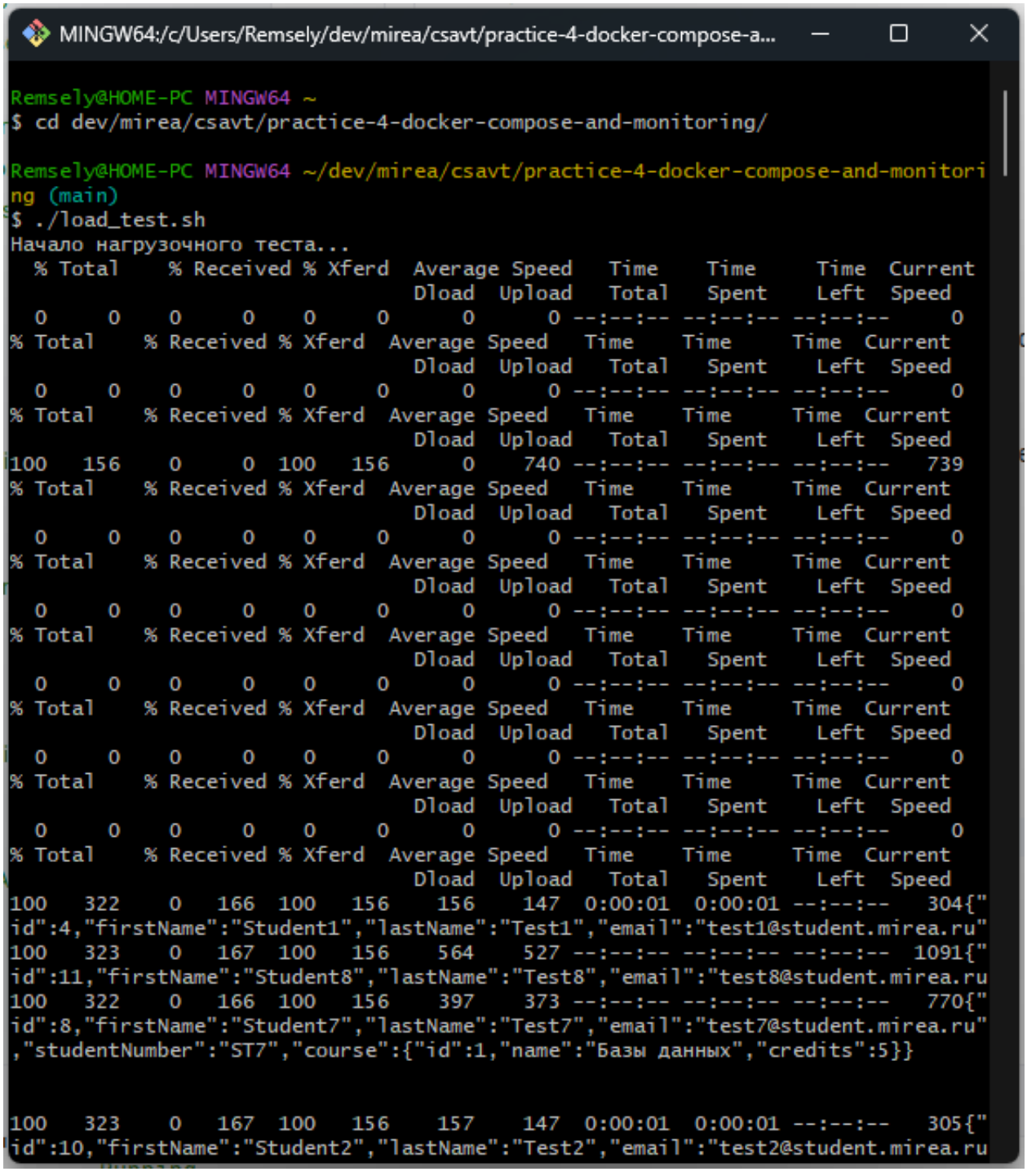

Рисунок 33 – Запуск скрипта для нагрузочного тестирования

Метрики приложения после нагрузочного тестирования представлены на рисунке 34.

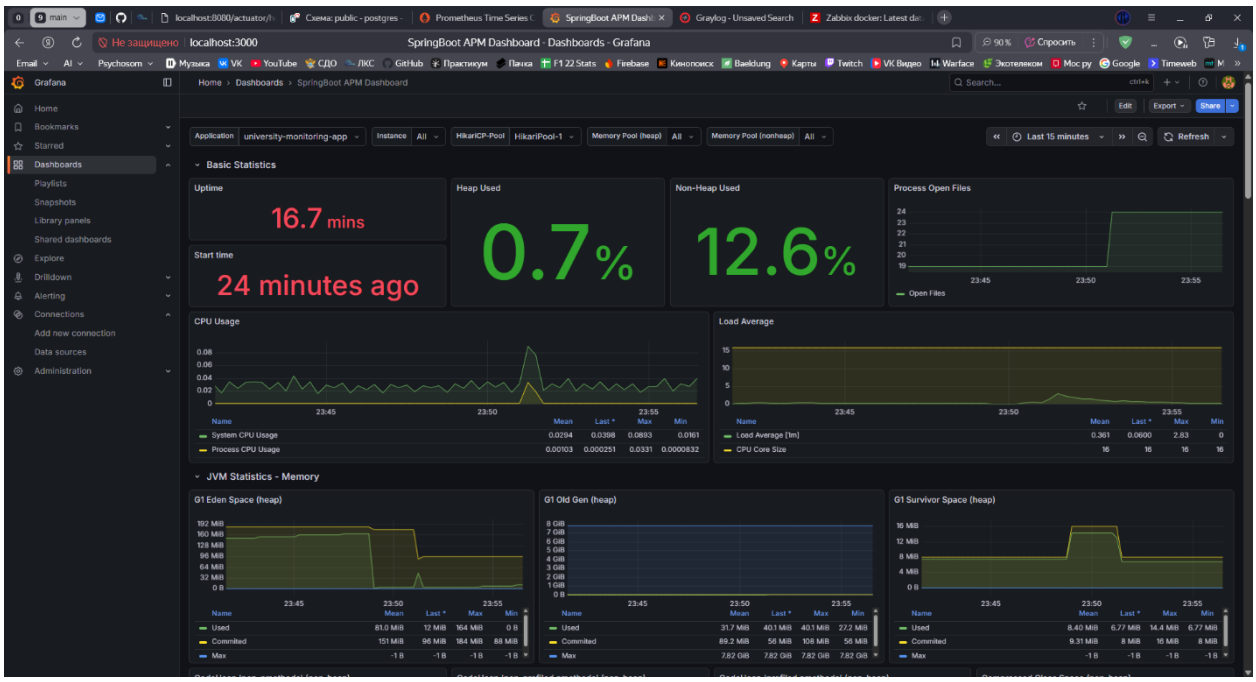

Рисунок 34 – Метрики приложения после нагрузочного тестирования

Вывод

В результате выполнения данной практической работы было разработано Spring Boot CRUD-приложение с окружением из инструментов мониторинга GrayLog, Adminer, Zabbix, Prometheus, Grafana.

Ответы на вопросы к практической работе

1. Назовите основные различия между Prometheus и Zabbix.

Prometheus использует модель pull, хранит метрики во временных рядах и ориентирован на cloud-native среду. Zabbix применяет агентов и push-модель, хранит данные в реляционной БД и подходит для классического инфраструктурного мониторинга.

2. Как можно запустить две базы PostgreSQL в одном docker-compose файле, чтобы таблицы не пересекались?

Нужно создать два сервиса PostgreSQL с разными именами, портами и томами. Каждый контейнер будет иметь свою БД и каталог данных, например: postgres1:5432, postgres2:5433, разные volumes и переменные окружения.

3. Назовите виды мониторинга систем.

Инфраструктурный, прикладной (APM), сетевой, лог-мониторинг, мониторинг безопасности, пользовательский и бизнес-мониторинг. Также выделяют три основы наблюдаемости: метрики, логи и трейсы.

4. При помощи чего можно передавать конфигурационные переменные в контейнер?

Через переменные окружения (environment или env_file), файлы конфигураций в volumes, Docker Secrets и Config, а также аргументы командной строки.

5. Назовите основные различия Docker Swarm и Docker Compose.

Docker Compose используется для локального запуска нескольких контейнеров. Docker Swarm — для кластерного развёртывания, масштабирования и балансировки нагрузки между нодами.

6. Назовите пример задачи, которую невозможно выполнить, используя один лишь docker-compose без Dockerfile.

Сборка собственного образа из исходников (например, компиляция приложения или установка зависимостей) требует Dockerfile; Compose может только запускать уже готовые образы.

Список источников информации

1. 50 вопросов по Docker, которые задают на собеседованиях, и ответы на них | Хабр. — Текст: электронный [сайт]. — URL: https://habr.com/ru/company/southbridge/blog/528206/
2. Docker Documentation | Docker Documentation — Текст: электронный [сайт]. — URL: — Текст: электронный [сайт]. — URL: https://docs.docker.com/
3. Zabbix Documentation — Текст: электронный [сайт]. — URL: https://www.zabbix.com/manuals
4. Prometheus Documentation — Текст: электронный [сайт]. — URL: https://prometheus.io/docs/introduction/overview/
5. Grafana Documentation — Текст: электронный [сайт]. — URL: https://grafana.com/docs/
6. GrayLog Documentation — Текст: электронный [сайт]. — URL: https://docs.graylog.org/