Департамент образования города Москвы

Государственное автономное образовательное учреждение

высшего образования города Москвы

«Московский городской педагогический университет»

Институт цифрового образования

Департамент информатики, управления и технологий

ДИСЦИПЛИНА:

«Интеграция и развертывание программного обеспечения с помощью контейнеров»

Лабораторная работа №3.1

Тема: «Docker Compose для мультиконтейнерных приложений»

Выполнила:

Студентка группы АДЭУ-211

Кравцова Алёна Евгеньевна

Руководитель:

Босенко Т.М

Москва

2024

Цель работы: освоить использование Docker Compose для управления многоконтейнерными приложениями.

Задачи:

1. Создать файл docker-compose.yml для указанного многоконтейнерного приложения.

2. Запустить приложение с помощью Docker Compose.

3. Проверить работоспособность приложения и взаимодействие между контейнерами.

4. Выполнить индивидуальное задание.

Индивидуальное задание:

st_93	Создать файл docker-compose.yml для системы учета сотрудников (Spring Boot + PostgreSQL).	Запустить приложение и проверить CRUD-операции.	Реализовать расчет средней зарплаты по отделам.

Шаг 1. Создание структуры проекта Spring Boot

Spring — это фреймворк для Java, на котором пишут веб-приложения и микросервисы. А Spring Boot — это расширение, которое упрощает и ускоряет работу со Spring. Оно представляет собой набор утилит, автоматизирующих настройки фреймворка.

Spring Boot разработан для ускорения создания веб-приложений. Он отличается от своего «родителя» тем, что не требует сложной настройки и имеет ряд встроенных инструментов, упрощающих написание кода.

Необходимо создать новый Spring Boot проект с зависимостями:

- Spring Web;
- Spring Data JPA;
- PostgreSQL Driver;

- Lombok;
- Micrometer Prometheus (для мониторинга).

Структура проекта:

employee-management/

- src/main/java/com/example/employees/

-- Employee.java

-- Department.java

-- EmployeeRepository.java

-- DepartmentRepository.java

-- EmployeeService.java

-- EmployeeController.java

-- DepartmentController.java

-- DataLoader.java

-- Application.java

- src/main/resources/application.yml

- pom.xml

- Dockerfile

- docker-compose.yml

- prometheus.yml

Структура представлена на рисунке 1.

Рис. 1 – Структура проекта

Шаг 2. Создание приложения

Теперь подробнее рассмотрим каждый структурный компонент проекта.

1) Employee.java

Employee.java – модель данных сотрудника (Рис. 2).

```java
package com.example.employees;

import jakarta.persistence.Entity;
import jakarta.persistence.Table;
import jakarta.persistence.Id;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.JoinColumn;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;

@Entity
@Table(name = "employees")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private Double salary;

    @ManyToOne
    @JoinColumn(name = "department_id")
    private Department department;
}
```

Рис. 2 – Employee.java

2) Department.java

Department.java – Модель данных отдела сотрудника (Рис. 3).

```java
package com.example.employees;

import jakarta.persistence.Entity;
import jakarta.persistence.Table;
import jakarta.persistence.Id;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;


@Entity
@Table(name = "departments")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Department {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
}
```

Рис. 3 – Department.java

3) EmployeeRepository.java

Подготовим хранилище данных о сотрудниках (Рис. 4).

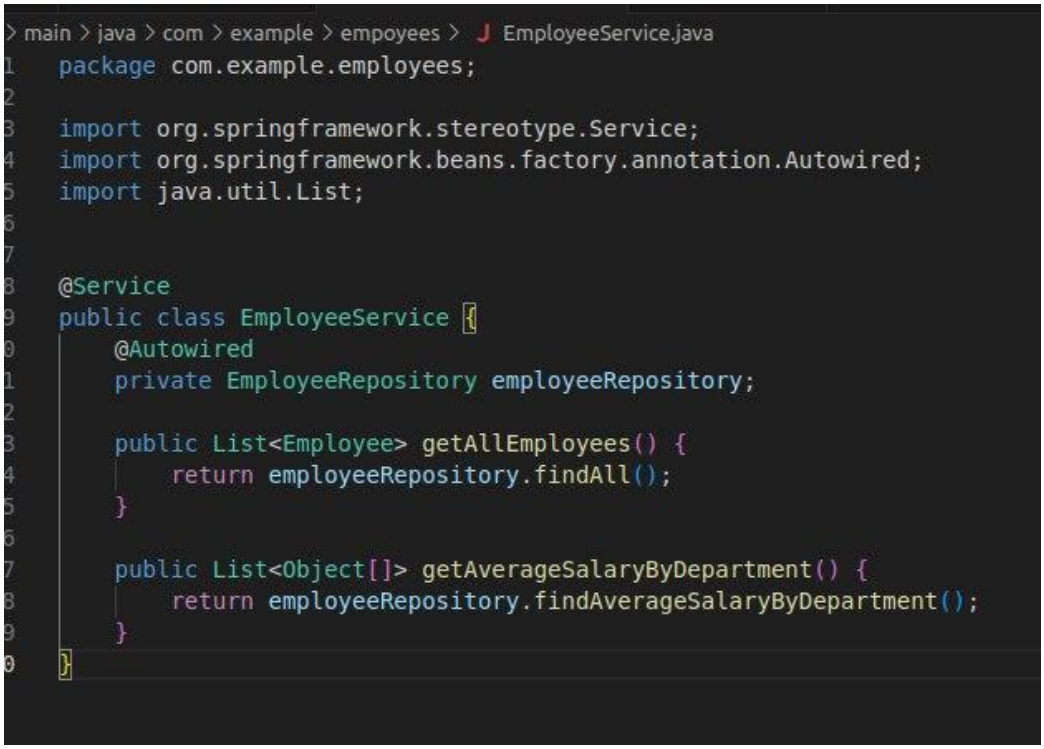


Рис. 4 – EmployeeRepository.java

4) DepartmentRepository.java

Подготовим хранилище данных о сотрудниках (Рис. 5).

Рис. 5 – DepartmentRepository.java

5) EmployeeService.java

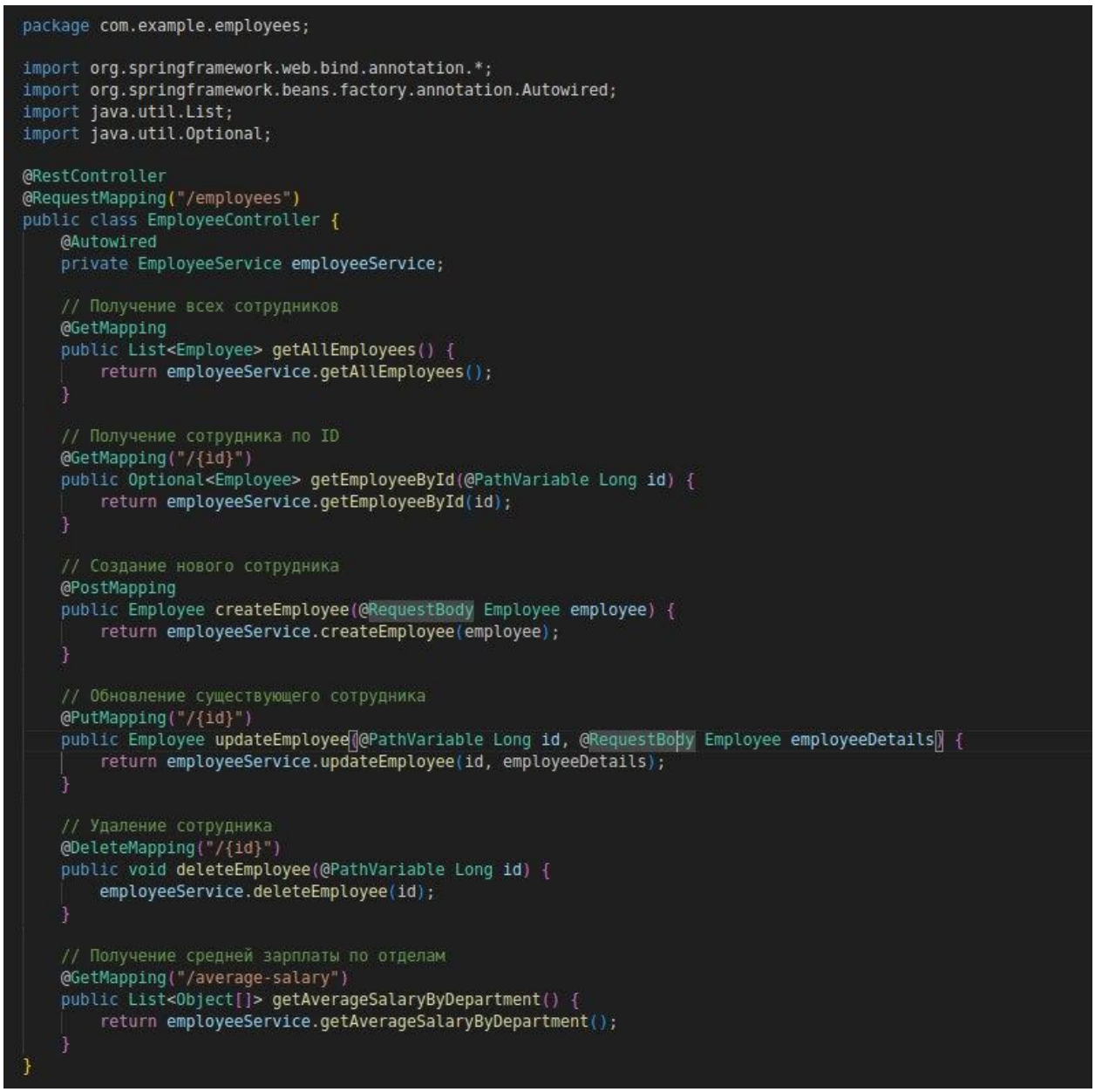
Сервис для получения всех сотрудников, а также расчета средней зарплаты по отделу (Рис. 6).

```java
package com.example.employees;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class EmployeeService {
    @Autowired
    private EmployeeRepository employeeRepository;

    // Получение всех сотрудников
    public List<Employee> getAllEmployees() {
        return employeeRepository.findAll();
    }

    // Получение сотрудника по ID
    public Optional<Employee> getEmployeeById(Long id) {
        return employeeRepository.findById(id);
    }

    // Создание нового сотрудника
    public Employee createEmployee(Employee employee) {
        return employeeRepository.save(employee);
    }

    // Обновление существующего сотрудника
    public Employee updateEmployee(Long id, Employee employeeDetails) {
        Employee employee = employeeRepository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Employee not found for this id :: " + id));

        employee.setName(employeeDetails.getName());
        employee.setSalary(employeeDetails.getSalary());
        employee.setDepartment(employeeDetails.getDepartment());
        return employeeRepository.save(employee);
    }

    // Удаление сотрудника
    public void deleteEmployee(Long id) {
        Employee employee = employeeRepository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Employee not found for this id :: " + id));
        employeeRepository.delete(employee);
    }

    // Получение средней зарплаты по отделам
    public List<Object[]> getAverageSalaryByDepartment() {
        return employeeRepository.findAverageSalaryByDepartment();
    }
}
```

Рис. 6 – EmployeeService.java

6) EmployeeController.java

Создание api для приложения с целью получения данных о сотрудниках (Рис. 7).



```java
package com.example.employees;

import org.springframework.web.bind.annotation.*;
import org.springframework.beans.factory.annotation.Autowired;
import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/employees")
public class EmployeeController {
    @Autowired
    private EmployeeService employeeService;

    // Получение всех сотрудников
    @GetMapping
    public List<Employee> getAllEmployees() {
        return employeeService.getAllEmployees();
    }

    // Получение сотрудника по ID
    @GetMapping("/{id}")
    public Optional<Employee> getEmployeeById(@PathVariable Long id) {
        return employeeService.getEmployeeById(id);
    }

    // Создание нового сотрудника
    @PostMapping
    public Employee createEmployee(@RequestBody Employee employee) {
        return employeeService.createEmployee(employee);
    }

    // Обновление существующего сотрудника
    @PutMapping("/{id}")
    public Employee updateEmployee(@PathVariable Long id, @RequestBody Employee employeeDetails) {
        return employeeService.updateEmployee(id, employeeDetails);
    }

    // Удаление сотрудника
    @DeleteMapping("/{id}")
    public void deleteEmployee(@PathVariable Long id) {
        employeeService.deleteEmployee(id);
    }

    // Получение средней зарплаты по отделам
    @GetMapping("/average-salary")
    public List<Object[]> getAverageSalaryByDepartment() {
        return employeeService.getAverageSalaryByDepartment();
    }
}
```

Рис. 7 – EmployeeController.java

7) DepartmentController.java

Создание api для приложения с целью получения данных об отделах (Рис. 8).

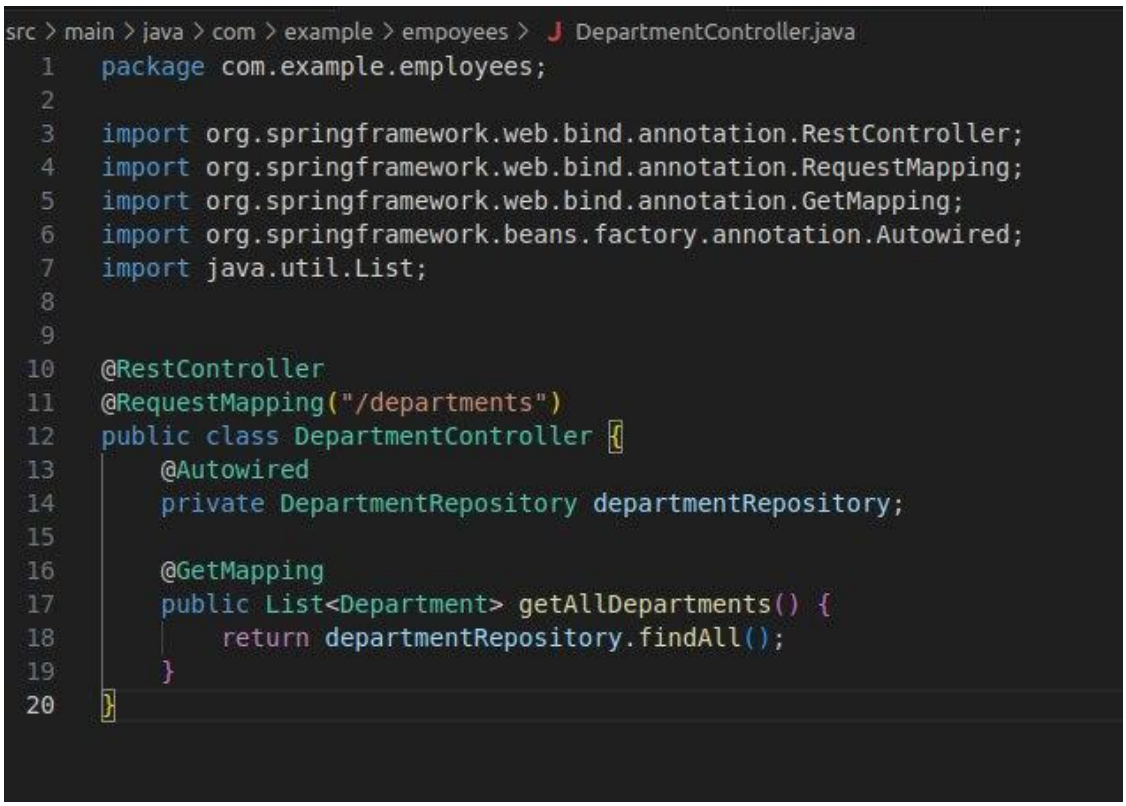


Рис. 8 – DepartmentController.java

8) DataLoader.java

DataLoader.java – файл, отвечающий за генерацию данных о сотрудниках (Рис. 9).

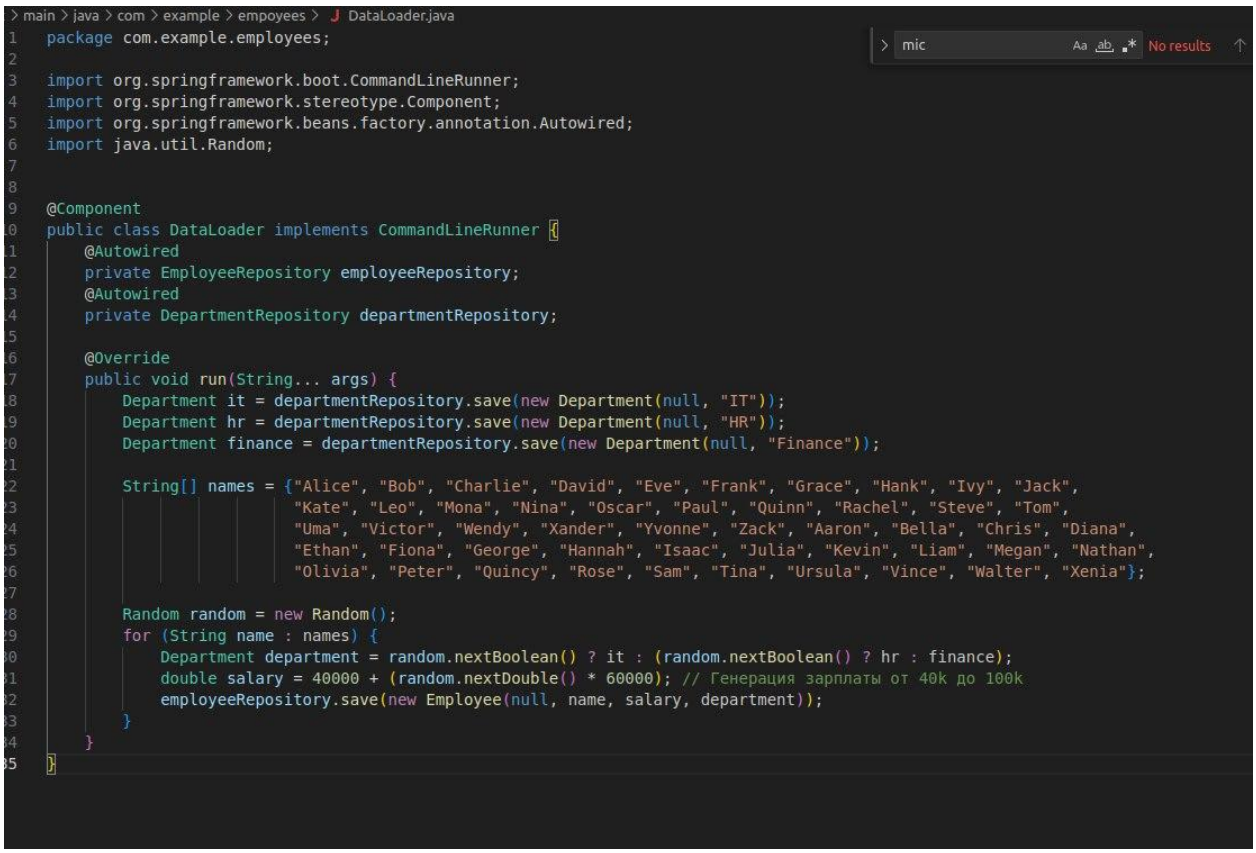


Рис. 9 – DataLoader.java

9) Application.java

Application.java – файл, запускающий приложение (Рис. 10).

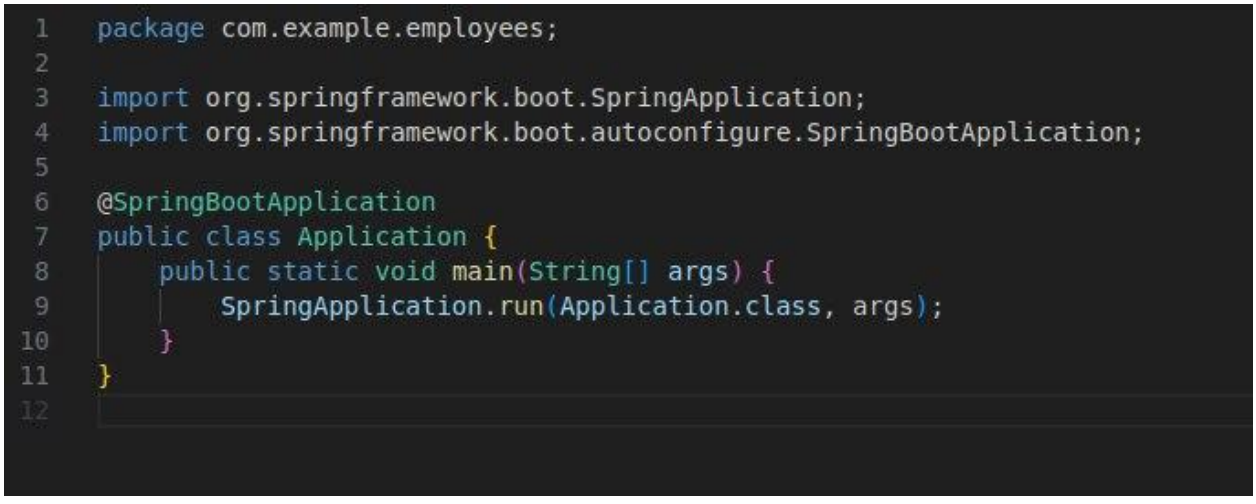
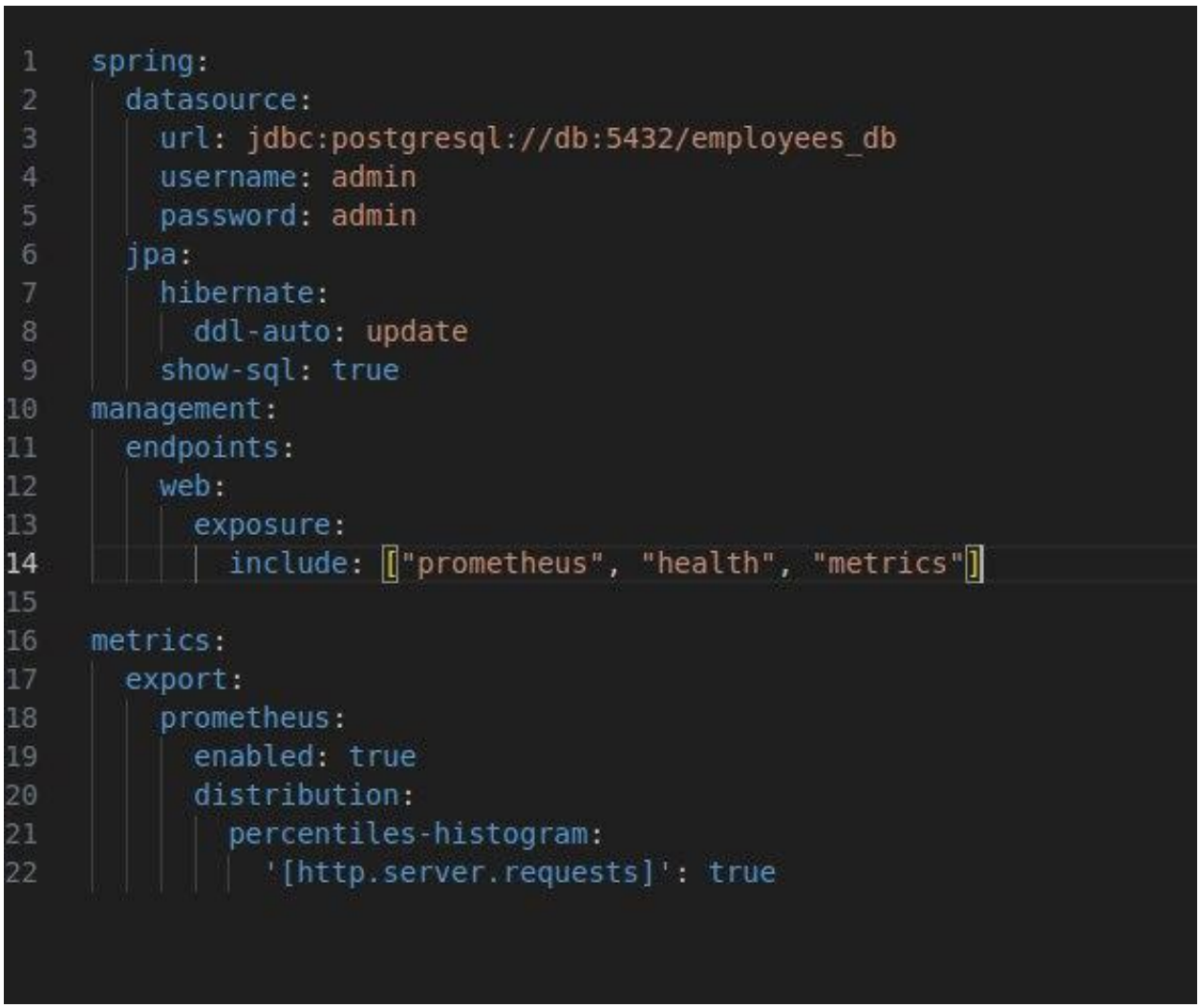


Рис. 10 – Application.java

10) application.yml

application.yml – файл, отвечающий за настройку приложения, а именно связывание его с БД и метриками Prometheus (Рис. 11).

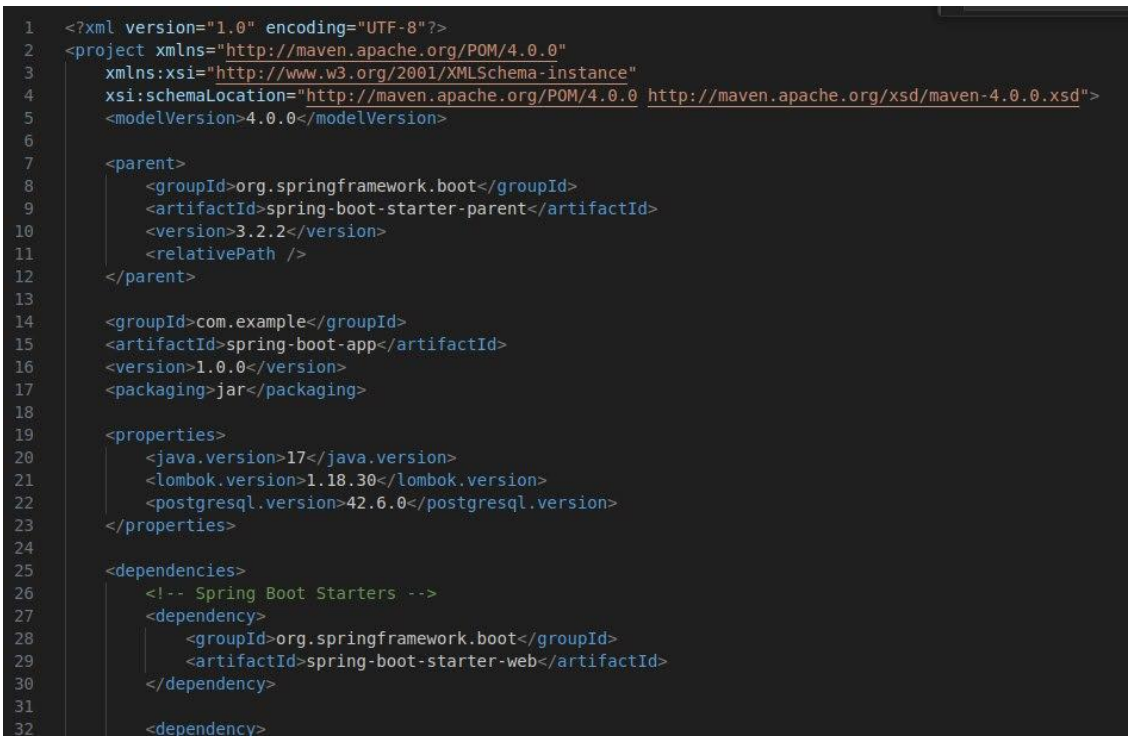


```yaml
spring:
  datasource:
    url: jdbc:postgresql://db:5432/employees_db
    username: admin
    password: admin
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
management:
  endpoints:
    web:
      exposure:
        include: ["prometheus", "health", "metrics"]

metrics:
  export:
    prometheus:
      enabled: true
      distribution:
        percentiles-histogram:
          '[http.server.requests]': true
```

Рис. 11 – application.yml

11) pom.xml

pom.xml – файл зависимости приложения (Рис. 12).

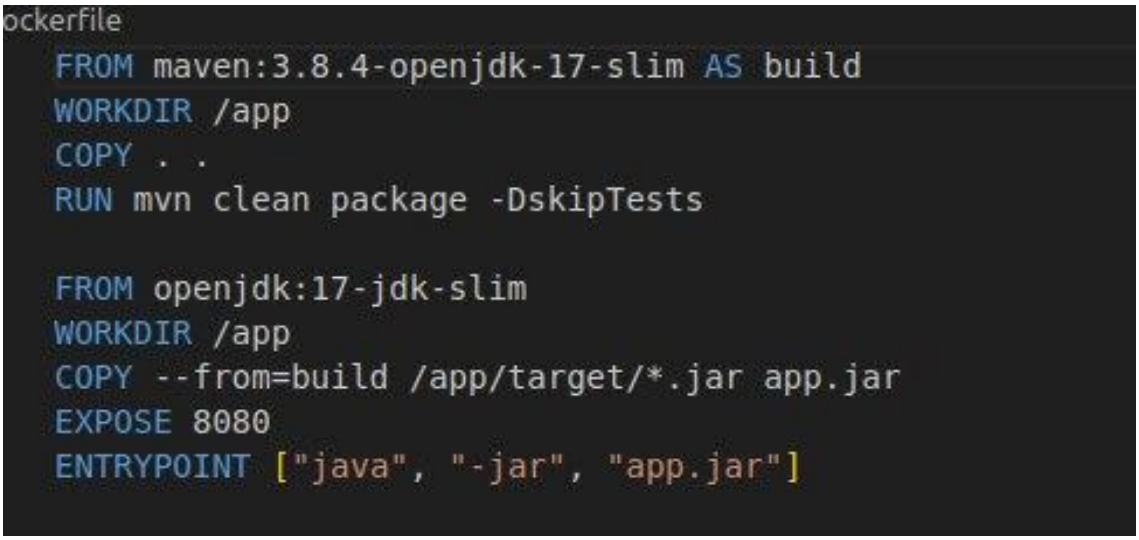


```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.2</version>
        <relativePath />
    </parent>

    <groupId>com.example</groupId>
    <artifactId>spring-boot-app</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <properties>
        <java.version>17</java.version>
        <lombok.version>1.18.30</lombok.version>
        <postgresql.version>42.6.0</postgresql.version>
    </properties>

    <dependencies>
        <!-- Spring Boot Starters -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
```

Рис. 12 – pom.xml

12) Dockerfile

Dockerfile – файл для конфигурирования приложения при запуске в Docker (Рис. 13).

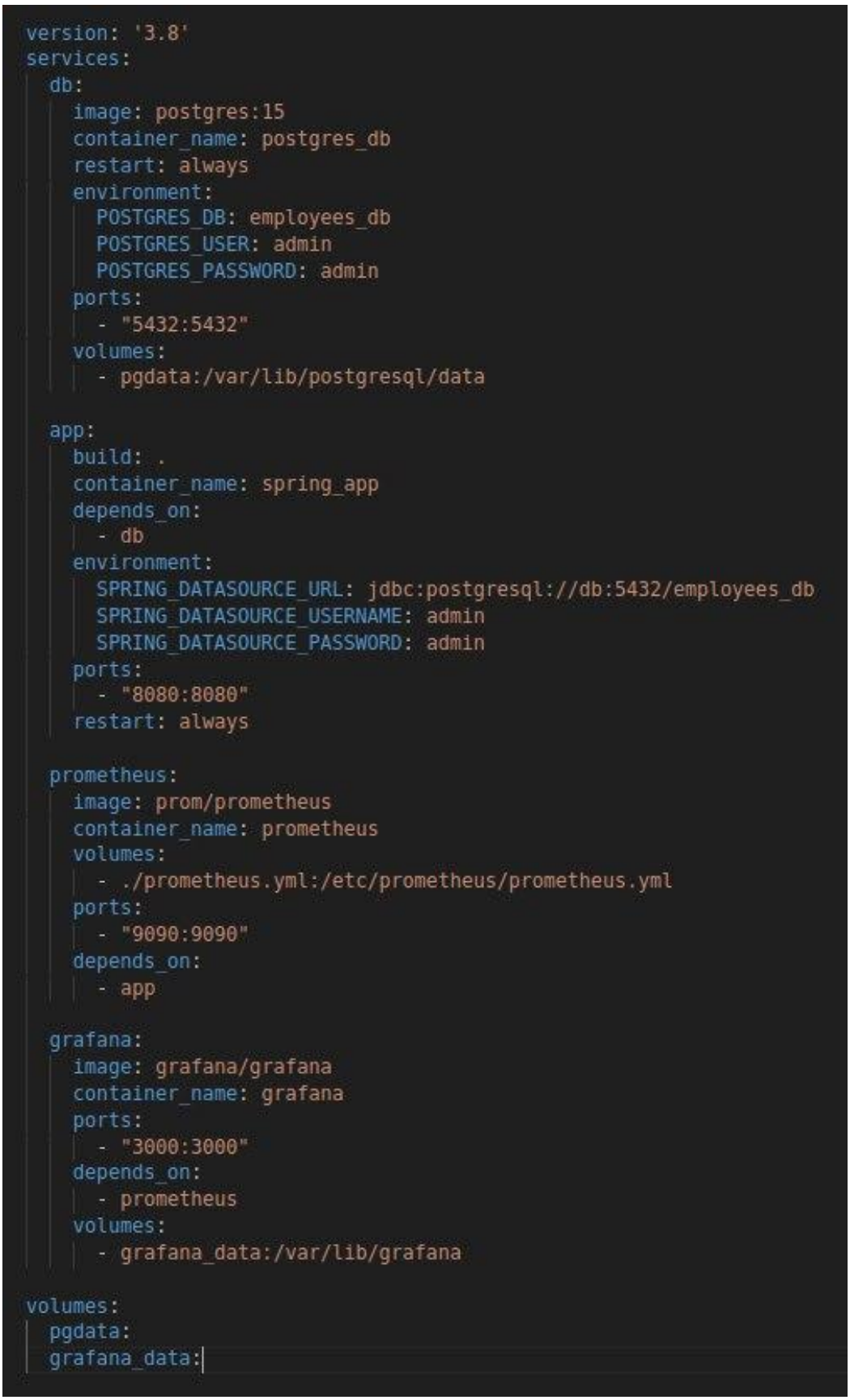


```
ockerfile
FROM maven:3.8.4-openjdk-17-slim AS build
WORKDIR /app
COPY . .
RUN mvn clean package -DskipTests

FROM openjdk:17-jdk-slim
WORKDIR /app
COPY --from=build /app/target/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Рис. 13 - Dockerfile

13) docker-compose.yml

Основной файл, который описывает 4-е контейнера и связь с БД, приложением, Prometheus и Grafana (Рис. 14).



```yaml
version: '3.8'
services:
  db:
    image: postgres:15
    container_name: postgres_db
    restart: always
    environment:
      POSTGRES_DB: employees_db
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: admin
    ports:
      - "5432:5432"
    volumes:
      - pgdata:/var/lib/postgresql/data

  app:
    build: .
    container_name: spring_app
    depends_on:
      - db
    environment:
      SPRING_DATASOURCE_URL: jdbc:postgresql://db:5432/employees_db
      SPRING_DATASOURCE_USERNAME: admin
      SPRING_DATASOURCE_PASSWORD: admin
    ports:
      - "8080:8080"
    restart: always

  prometheus:
    image: prom/prometheus
    container_name: prometheus
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
      - "9090:9090"
    depends_on:
      - app

  grafana:
    image: grafana/grafana
    container_name: grafana
    ports:
      - "3000:3000"
    depends_on:
      - prometheus
    volumes:
      - grafana_data:/var/lib/grafana

volumes:
  pgdata:
  grafana_data:
```

Рис. 14 – docker-compose.yml

14) prometheus.yml

Файл конфигурации Prometheus с приложением (Рис. 15).

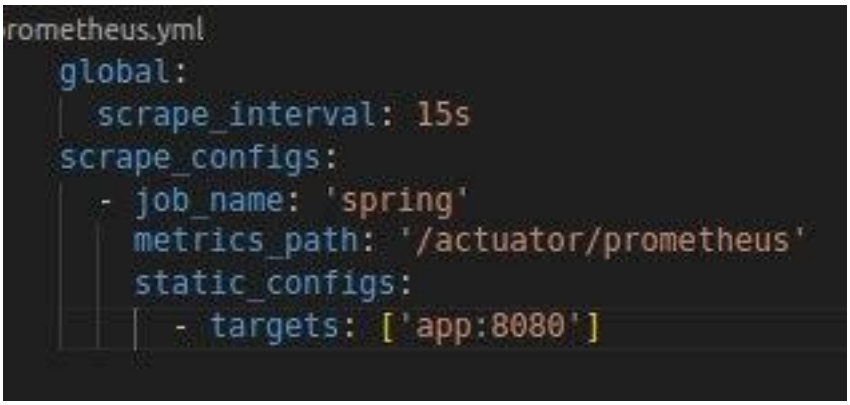


Рис. 15 – prometheus.yml

Шаг 3. Запуск приложения

Сделаем *Docker compose build up –d –build* для сборки контейнеров и запуска приложения (Рис. 16).

Рис. 16 – Сборка контейнера

Через *docker ps* посмотрим запущенные контейнеры (Рис. 17).

Рис. 17 – Просмотр всех запущенных контейнеров

Исходя из рисунка 17 видно, что все контейнеры успешно запущены и работают.

Шаг 4. Проверка CRUD операций

Read: выводим всех сотрудников (Рис. 19).

Рис. 18 – read всех сотрудников

Read: выводим среднюю ЗП по отделу (Рис. 19).

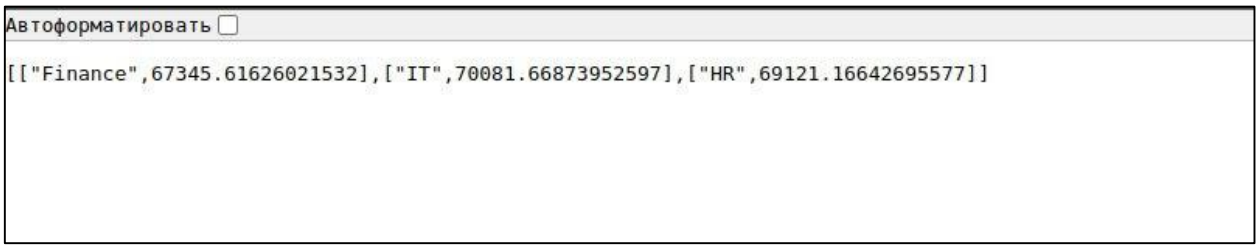


Рис. 19 – read средней ЗП

Delete: удаление сотрудника (Рис. 20).

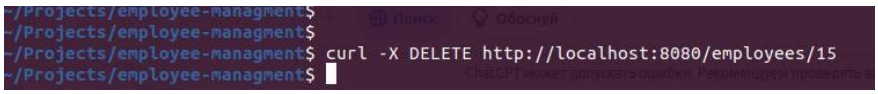


Рис. 20 – Delete

Put: обновление данных о сотруднике (Рис. 21).

Рис. 21 – Put

Шаг 5. Настройка и визуализация посредством Prometheus и Grafana

Проверим связь Prometheus и созданного приложения (Рис. 22).

Рис. 22 – Проверка взаимодействия

Далее выведем количества HTTP-запросов, обработанных сервером, с момента его последнего запуска (Рис. 23).

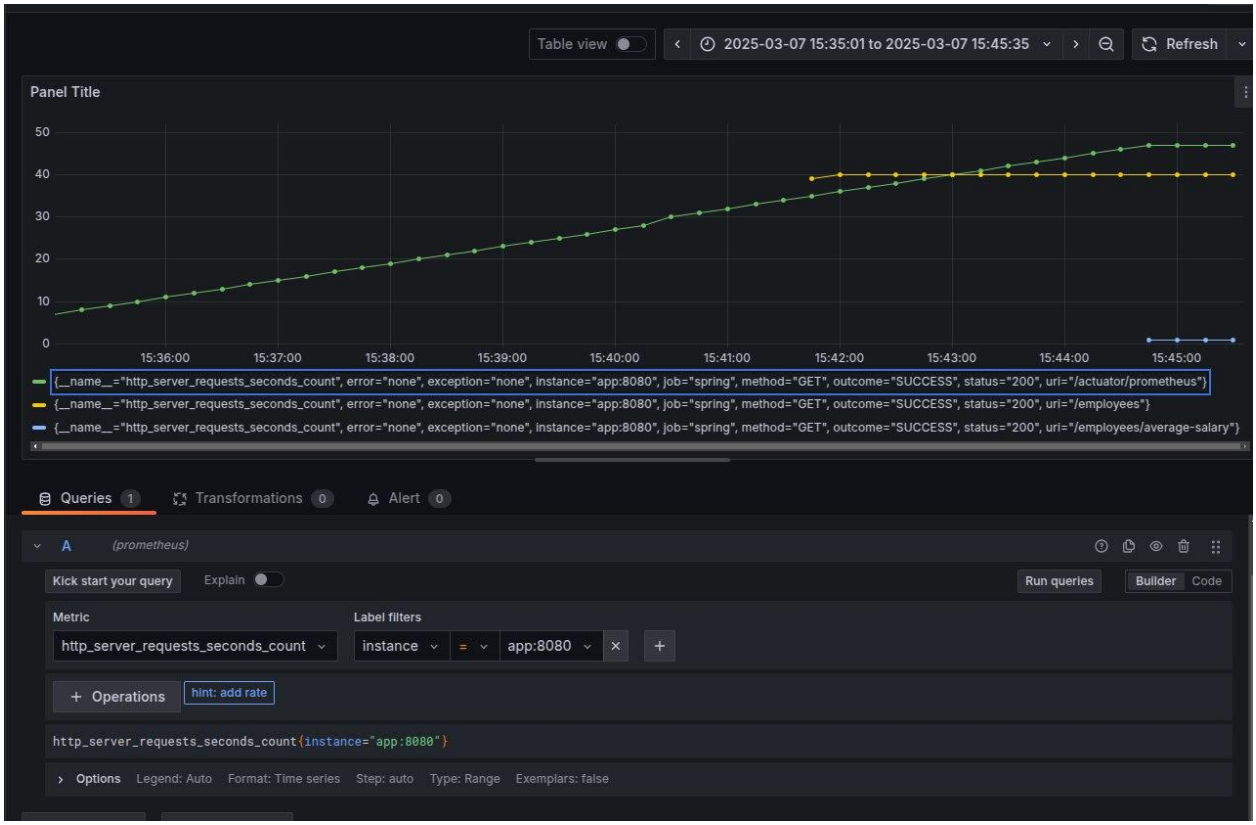


Рис. 23 – Количество HTTP-запросов

Далее с помощью Grafana выведем данные запроса http://localhost:8080/employees/average-salary по средней ЗП по отделу (Рис. 24).

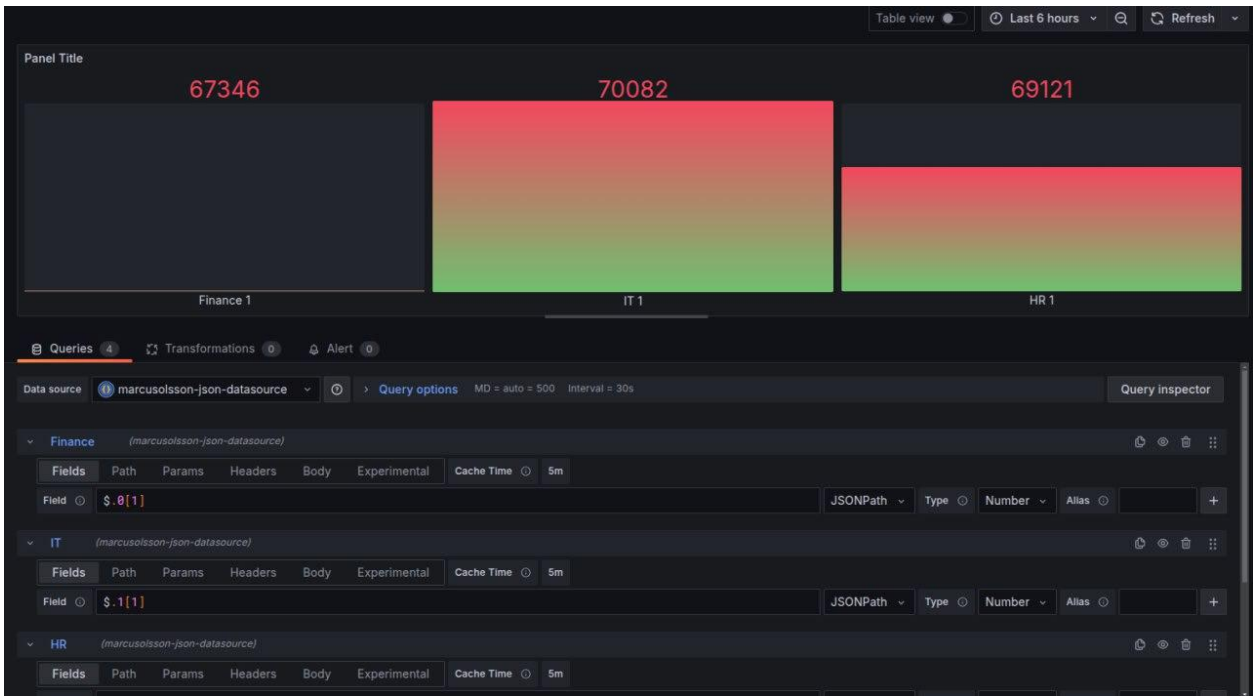


Рис. 24 – Дашборд о средней ЗП по отделу

Заключение: таким образом, в ходе выполнения работы было создано многоконтейнерное приложение на spring boot с 4-мя контейнерами. Были сгенерированы данные, успешно загружены в БД, а также подключен Prometheus и Grafana. Все задачи были выполнены.