|  |
| --- |
|  |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |
| Институт информационных технологий |
| Кафедра инструментального и прикладного программного обеспечения |

|  |  |
| --- | --- |
|  | |
| **ОТЧЕТ ПО ПРАКТИЧЕСКИМ РАБОТАМ № 1 — 24** | |
| **по дисциплине** | |
| **«Шаблоны программных платформ языка Джава»** | |
| Выполнил студент группы **ИКБО-20-22** | ***Шумахер М.Е.*** |
| Принял ассистент кафедры ИиППО | ***Ермаков С.Р.*** |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | « » 2024г. |  |
| «Зачтено» | « » 2024г. |  |

Москва 2024

**ОГЛАВЛЕНИЕ**

[Практическая работа №1 3](#_Toc167322524)

[Практическая работа №2 4](#_Toc167322525)

[Практическая работа №3 6](#_Toc167322526)

[Практическая работа №4 9](#_Toc167322527)

[Практическая работа №5 11](#_Toc167322528)

[Практическая работа №6 12](#_Toc167322529)

[Практическая работа №7 18](#_Toc167322530)

[Практическая работа №8 21](#_Toc167322531)

[Практическая работа №9 23](#_Toc167322532)

[Практическая работа №10 24](#_Toc167322533)

[Практическая работа №11 26](#_Toc167322534)

[Практическая работа №12 27](#_Toc167322535)

[Практическая работа №13 29](#_Toc167322536)

[Практическая работа №14 31](#_Toc167322537)

[Практическая работа №15 33](#_Toc167322538)

[Практическая работа №16 38](#_Toc167322539)

[Практическая работа №17 46](#_Toc167322540)

[Практическая работа №18 51](#_Toc167322541)

[Практическая работа №19 54](#_Toc167322542)

[Практическая работа №20 56](#_Toc167322543)

[Практическая работа №21 57](#_Toc167322544)

[Практическая работа №22 58](#_Toc167322545)

[Практическая работа №23 59](#_Toc167322546)

[Практическая работа №24 63](#_Toc167322547)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 66](#_Toc167322548)

Практическая работа №1

1. **Постановка задачи и персональный вариант**

Цель: Знакомство со встроенными функциональными интерфейсами Java. Возможности Java 8. Лямбда-выражения. Области действия, замыкания. Предикаты. Функции. Компараторы.

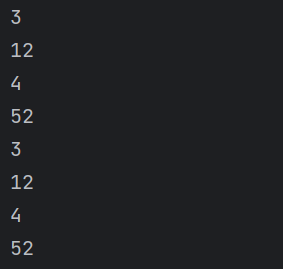
Постановка задачи: имплементировать интерфейс Consumer, принимающий на вход массив чисел и выводящий в консоль в порядке возрастания.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac1>

package prac1;  
  
import java.util.function.Consumer;  
  
public class Main implements Consumer<int[]> {  
 @Override  
 public void accept(int[] num){  
 for (int n : num){  
 System.*out*.println(n + " ");  
 }  
 }  
 public static void main(String[] args) {  
 int[] num ={3,12,4,52};  
 Main cons = new Main();  
 cons.accept(num);  
 Consumer<int[]> printer = arr -> {  
 for (int n : arr) {  
 System.*out*.println(n + " ");  
 }  
 };  
 printer.accept(num);  
 }  
}

1. **Результат работы программы**



Практическая работа №2

1. **Постановка задачи и персональный вариант**

Цель: Работа со Stream API в Java 8

Постановка задачи: Увеличение веса каждого объекта на 3, сортировка по весу в обратном порядке, фильтрация по дате рождения меньшей, чем 01.01.2000, сумма всех весов.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac2>

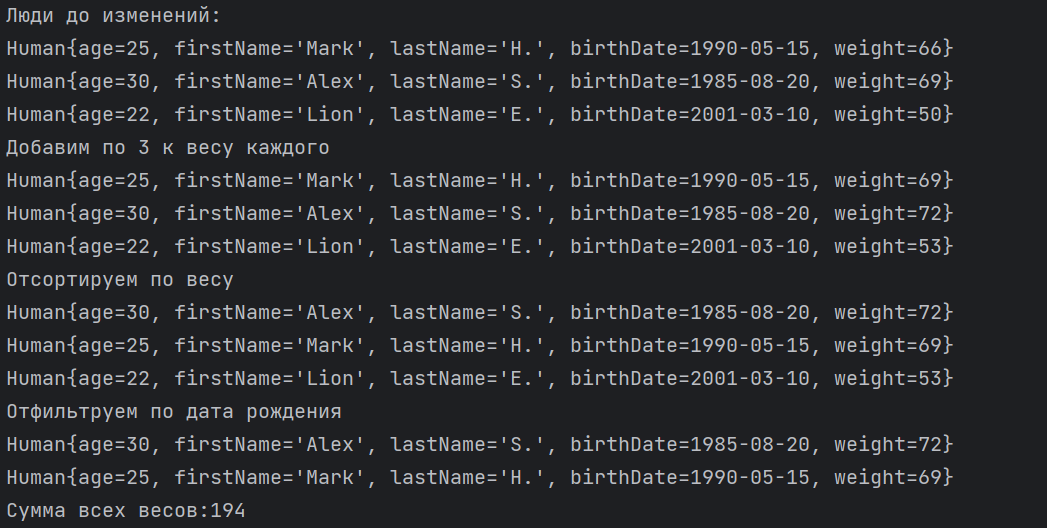
Класс Human:

package prac2;  
  
import java.time.LocalDate;  
  
public class Human {  
 private int age;  
 private String firstName;  
 private String lastName;  
 private LocalDate birthDate;  
 private int weight;  
  
 public Human(int age, String firstName, String lastName, LocalDate birthDate, int weight) {  
 this.age = age;  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.birthDate = birthDate;  
 this.weight = weight;  
 }  
  
 public int getAge() {  
 return age;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public LocalDate getBirthDate() {  
 return birthDate;  
 }  
  
 public int getWeight() {  
 return weight;  
 }  
  
 public void setWeight(int weight) {  
 this.weight = weight;  
 }  
  
 @Override  
 public String toString() {  
 return "Human{" +  
 "age=" + age +  
 ", firstName='" + firstName + '\'' +  
 ", lastName='" + lastName + '\'' +  
 ", birthDate=" + birthDate +  
 ", weight=" + weight +  
 '}';  
 }  
}

Класс Main:

package prac2;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.stream.Collectors;  
  
public class Main {  
 public static void main(String[] args) {  
 List<Human> h\_list = new ArrayList<>();  
 h\_list.add(new Human(25, "Mark", "H.", LocalDate.*of*(1990, 5, 15), 66));  
 h\_list.add(new Human(30, "Alex", "S.", LocalDate.*of*(1985, 8, 20), 69));  
 h\_list.add(new Human(22, "Lion", "E.", LocalDate.*of*(2001, 3, 10), 50));  
 System.*out*.println("Люди до изменений:");  
 h\_list.forEach(System.*out*::println);  
 System.*out*.println("Добавим по 3 к весу каждого");  
 h\_list = h\_list.stream()  
 .map(human -> {  
 human.setWeight(human.getWeight()+3);  
 return human;  
 }).collect(Collectors.*toList*());  
 h\_list.forEach(System.*out*::println);  
  
  
 System.*out*.println("Отсортируем по весу");  
 h\_list=h\_list.stream().sorted((human1,human2) -> human2.getWeight() - human1.getWeight()).collect(Collectors.*toList*());  
 h\_list.forEach(System.*out*::println);  
  
  
 System.*out*.println("Отфильтруем по дата рождения");  
 h\_list.stream().filter(human -> human.getBirthDate().isBefore(LocalDate.*of*(2000,1,1))).forEach(System.*out*::println);  
  
  
 int sum = h\_list.stream().mapToInt(Human::getWeight).sum();  
 System.*out*.println("Сумма всех весов:" + sum);  
 }  
}

1. **Результат работы программы**



**Практическая работа №3**

1. **Постановка задачи и персональный вариант**

Тема: знакомство с конкурентным программированием в Java. Потокобезопасность, ключевое слово syncrhonized, мьютексы, семафоры, мониторы, барьеры.

Постановка задачи: создать свои потокобезопасные имплементации интерфейсов: Map с использованием Lock, Set с использованием Semaphore.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac3>

package prac3.part1;  
  
public class Main {  
 public static void main(String[] args) {  
 ConcurrentMap<String, Integer> concurrentMap = new ConcurrentMap<>();  
  
 Thread thread1 = new Thread(() -> {  
 try {  
 for (int i = 0; i < 5; i++) {  
 concurrentMap.put("Key" + i, i);  
 System.*out*.println("Thread 1 added: Key" + i + " -> " + i);  
 Thread.*sleep*(1000);  
 }  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 });  
  
 Thread thread2 = new Thread(() -> {  
 try {  
 for (int i = 5; i < 10; i++) {  
 concurrentMap.put("Key" + i, i);  
 System.*out*.println("Thread 2 added: Key" + i + " -> " + i);  
 Thread.*sleep*(1000);  
 }  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 });  
  
 thread1.start();  
 thread2.start();  
 }  
}

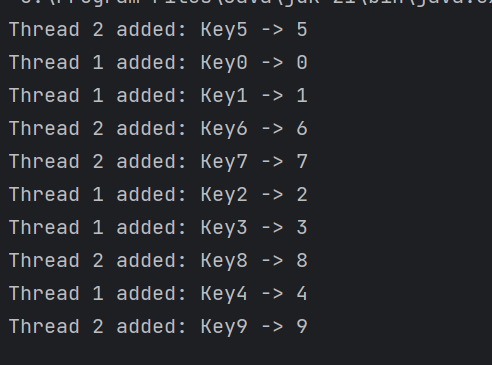
package prac3.part1;  
  
import java.util.HashMap;  
import java.util.Map;  
import java.util.concurrent.locks.Lock;  
import java.util.concurrent.locks.ReentrantLock;  
  
public class ConcurrentMap<K, V> {  
 private final Map<K, V> map;  
 private final Lock lock;  
  
 public ConcurrentMap() {  
 map = new HashMap<>();  
 lock = new ReentrantLock();  
 }  
  
 public void put(K key, V value) {  
 lock.lock();  
 try {  
 map.put(key, value);  
 } finally {  
 lock.unlock();  
 }  
 }  
}

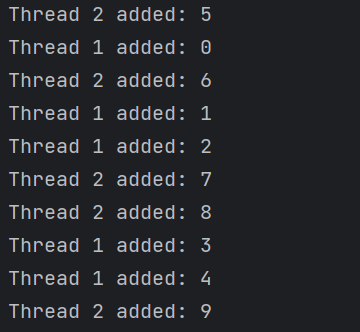
2-я часть:

package prac3;  
  
import java.util.HashSet;  
import java.util.Set;  
import java.util.concurrent.Semaphore;  
  
public class ConcurrentSet<T> {  
 private Set<T> set;  
 private Semaphore semaphore;  
  
 public ConcurrentSet() {  
 set = new HashSet<>();  
 semaphore = new Semaphore(1); // Одна разрешающая семафор  
 }  
  
 public boolean add(T element) throws InterruptedException {  
 semaphore.acquire(); // Захватываем разрешение  
 boolean added = set.add(element);  
 semaphore.release(); // Освобождаем разрешение  
 return added;  
 }  
}

package prac3;  
  
public class Main {  
 public static void main(String[] args) {  
 ConcurrentSet<Integer> concurrentSet = new ConcurrentSet<>();  
  
 Thread thread1 = new Thread(() -> {  
 try {  
 for (int i = 0; i < 5; i++) {  
 concurrentSet.add(i);  
 System.*out*.println("Thread 1 added: " + i);  
 Thread.*sleep*(1000);  
 }  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 });  
  
 Thread thread2 = new Thread(() -> {  
 try {  
 for (int i = 5; i < 10; i++) {  
 concurrentSet.add(i);  
 System.*out*.println("Thread 2 added: " + i);  
 Thread.*sleep*(1000);  
 }  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 });  
  
 thread1.start();  
 thread2.start();  
 }  
}

1. **Результат работы программы**





**Практическая работа №4**

1. **Постановка задачи и персональный вариант**

Цель: работа с ExecutorService, CompletableFuture

Постановка задачи: реализовать собственную имплементацию ExecutorService с единственным параметром конструктора – количеством потоков.

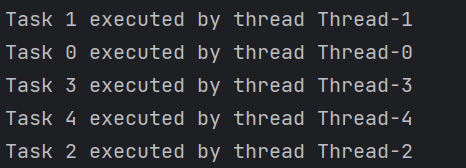
1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac4>

package prac4;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.concurrent.BlockingQueue;  
import java.util.concurrent.LinkedBlockingQueue;  
  
class CustomExecutorService  
{  
 private final List<WorkerThread> threads;  
 private final BlockingQueue<Runnable> taskQueue;  
  
 public CustomExecutorService(int threadPoolSize)  
 {  
 this.threads = new ArrayList<>(threadPoolSize);  
 this.taskQueue = new LinkedBlockingQueue<>();  
  
 for (int i = 0; i < threadPoolSize; i++)  
 {  
 WorkerThread workerThread = new WorkerThread();  
 threads.add(workerThread);  
 workerThread.start();  
 }  
 }  
  
 public void submit(Runnable task)  
 {  
 try  
 {  
 taskQueue.put(task);  
 }  
 catch (InterruptedException e)  
 {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
  
 public void shutdown()  
 {  
 for (WorkerThread workerThread : threads)  
 {  
 workerThread.interrupt();  
 }  
 }  
  
 private class WorkerThread extends Thread  
 {  
 @Override  
 public void run()  
 {  
 while (!Thread.*currentThread*().isInterrupted())  
 {  
 try  
 {  
 Runnable task = taskQueue.take();  
 task.run();  
 }  
 catch (InterruptedException e)  
 {  
 Thread.*currentThread*().interrupt();  
 }  
 }  
 }  
 }  
}

package prac4;  
  
public class Main  
{  
 public static void main(String[] args)  
 {  
 CustomExecutorService executorService = new CustomExecutorService(5);  
  
 for (int i = 0; i < 10; i++)  
 {  
 final int taskNumber = i;  
 executorService.submit(() -> {  
 System.*out*.println("Task " + taskNumber + " executed by thread " + Thread.*currentThread*().getName());  
 });  
 }  
  
 executorService.shutdown();  
 }  
}

1. **Результат работы программы**



**Практическая работа №5**

1. **Постановка задачи и персональный вариант**

Цель: познакомиться с паттернами проектирования, их определением и классификацией. Обзор паттернов GoF. Паттерн Синглтон.

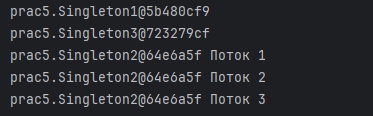
Постановка задачи: реализовать паттерн Singleton как минимум 3-мя способами.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac5>

package prac5;  
  
public class Main {  
 public static void main(String[] args) {  
 System.*out*.println(Singleton1.*getInstance*());  
 System.*out*.println(Singleton3.*getInstance*());  
 for (int i = 1; i < 4; i++) {  
 var t = new Thread(() -> {  
 System.*out*.println(Singleton2.*getInstance*() + " " + Thread.*currentThread*().getName());  
 });  
 t.setName("Поток " + i);  
 t.start();  
 }  
 }  
}  
  
class Singleton1{//ленивая инициализация  
 private static Singleton1 *instance*;  
 private Singleton1(){}  
 public static Singleton1 getInstance(){  
 if (*instance* == null){  
 *instance* = new Singleton1();  
 }  
 return *instance*;  
 }  
}  
class Singleton2 {//потокобезопасный  
 private static Singleton2 *instance*;  
 private Singleton2(){}  
 public static synchronized Singleton2 getInstance(){  
 if (*instance* == null){  
 *instance* = new Singleton2();  
 }  
 return *instance*;  
 }  
}  
class Singleton3 {//холодная инициализация  
 private static final Singleton3 *instance* = new Singleton3();  
 private Singleton3() {}  
 public static Singleton3 getInstance() {  
 return *instance*;  
 }  
}

1. **Результат работы программы**



**Практическая работа №6**

1. **Постановка задачи и персональный вариант**

Цель: знакомство с реализацией порождающих паттернов проектирования.

Постановка задачи: написать реализацию паттернов «Фабричный метод», «Абстрактная фабрика», «Строитель», «Прототип».

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac6>

package prac6;  
  
public class Main {  
 public static void main(String[] args) {  
 // «Фабричный метод»  
 System.*out*.println("Паттерн «Фабричный метод»:");  
 new FactoryMethod();  
 System.*out*.println();  
  
 // «Абстрактная фабрика»  
 System.*out*.println("Паттерн «Абстрактная фабрика»:");  
 new AbstractFactory();  
 System.*out*.println();  
  
 // «Строитель»  
 System.*out*.println("Паттерн «Строитель»:");  
 new Builder();  
 System.*out*.println();  
  
 // «Прототип»  
 System.*out*.println("Паттерн «Прототип»:");  
 new Prototype();  
  
 }  
}

import java.util.List;  
import java.util.concurrent.BlockingQueue;  
import java.util.concurrent.LinkedBlockingQueue;

}  
}

FactoryMethod

package prac6;  
  
public class FactoryMethod{  
 public FactoryMethod() {  
 Work();  
 }  
  
 private void Work() {  
 Factory factory = new PanelFactory("Завод панельных домов");  
 factory.BuildHouse();  
 factory = new WoodFactory("Завод деревянных домов");  
 factory.BuildHouse();  
 }  
}  
  
// Фабрики  
abstract class Factory{  
 public String Name;  
  
 public Factory(String name) {  
 Name = name;  
 }  
  
 abstract House BuildHouse();  
}  
  
class PanelFactory extends Factory{  
 public PanelFactory(String name) {  
 super(name);  
 }  
  
 @Override  
 House BuildHouse() {  
 return new PanelHouse();  
 }  
}  
  
class WoodFactory extends Factory {  
 public WoodFactory(String name) {  
 super(name);  
 }  
  
 @Override  
 House BuildHouse() {  
 return new WoodHouse();  
 }  
}  
  
// Виды домов  
interface House { }  
  
class PanelHouse implements House {  
 public PanelHouse() {  
 System.*out*.println("Panel house!");  
 }  
}  
  
class WoodHouse implements House{  
 public WoodHouse() {  
 System.*out*.println("Wooden house!");  
 }  
}

AbstractFactory

package prac6;  
  
public class AbstractFactory{  
 public AbstractFactory(){  
 Work();  
 }  
  
 private void Work(){  
 var hero1 = new Hero(new ElfFactory());  
 var hero2 = new Hero(new TankFactory());  
 hero1.Attack();  
 hero2.Attack();  
 }  
}  
  
// Оружие  
abstract class Weapon{  
 abstract void Hit();  
}  
class Sword extends Weapon{  
 public void Hit(){  
 System.*out*.println("Удар мечом!");  
 }  
}  
class Bow extends Weapon{  
 public void Hit(){  
 System.*out*.println("Выстрел из лука!");  
 }  
}  
  
// Герой  
class Hero{  
 private Weapon HeroWeapon;  
  
 public Hero(HeroFactory factory){  
 HeroWeapon = factory.CreateWeapon();  
 }  
  
 public void Attack(){  
 HeroWeapon.Hit();  
 }  
}  
  
// Фабрики  
abstract class HeroFactory{  
 abstract Weapon CreateWeapon();  
}  
class TankFactory extends HeroFactory{  
 public Weapon CreateWeapon(){  
 return new Sword();  
 }  
}  
class ElfFactory extends HeroFactory{  
 public Weapon CreateWeapon(){  
 return new Bow();  
 }  
}

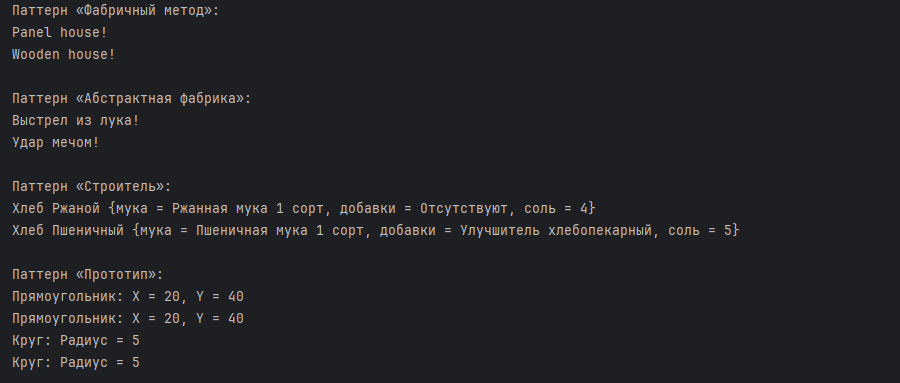
Builder

package prac6;  
  
public class Builder{  
 public Builder(){  
 Work();  
 }  
  
 private void Work(){  
 var baker = new Baker();  
 var bread1 = baker.CreateBread(new RyeBreadBuilder());  
 System.*out*.println(bread1);  
 var bread2 = baker.CreateBread(new WheatBreadBuilder());  
 System.*out*.println(bread2);  
 }  
}  
  
// Для хлеба  
class Flour{  
 public String FlourType;  
  
 public Flour(String flourType){  
 FlourType = flourType;  
 }  
}  
class Additive{  
 public String AdditiveName;  
  
 public Additive(String additiveName){  
 AdditiveName = additiveName;  
 }  
}  
class Salt{  
 public int saltCount;  
  
 public Salt(int count){  
 saltCount = count;  
 }  
}  
  
class Bread{  
 public String name;  
 public Flour flour;  
 public Additive additive;  
 public Salt salt;  
  
 @Override  
 public String toString(){  
 return "Хлеб " + name + " {" +  
 "мука = " + flour.FlourType +  
 ", добавки = " + additive.AdditiveName +  
 ", соль = " + salt.saltCount + "}";  
 }  
}  
  
// Получение хлеба  
class Baker{  
 public Bread CreateBread(BreadBuilder builder){  
 builder.SetName();  
 builder.SetFlour();  
 builder.SetAdditive();  
 builder.SetSalt();  
 return builder.bread;  
 }  
}  
  
abstract class BreadBuilder{  
 public Bread bread;  
  
 public BreadBuilder(){  
 bread = new Bread();  
 }  
  
 abstract void SetName();  
 abstract void SetFlour();  
 abstract void SetAdditive();  
 abstract void SetSalt();  
}  
  
class RyeBreadBuilder extends BreadBuilder{  
 void SetName(){  
 this.bread.name = "Ржаной";  
 }  
 void SetFlour(){  
 this.bread.flour = new Flour("Ржанная мука 1 сорт");  
 }  
 void SetAdditive(){  
 this.bread.additive = new Additive("Отсутствуют");  
 }  
 void SetSalt(){  
 this.bread.salt = new Salt(4);  
 }  
}  
class WheatBreadBuilder extends BreadBuilder{  
 void SetName(){  
 this.bread.name = "Пшеничный";  
 }  
 void SetFlour(){  
 this.bread.flour = new Flour("Пшеничная мука 1 сорт");  
 }  
 void SetAdditive(){  
 this.bread.additive = new Additive("Улучшитель хлебопекарный");  
 }  
 void SetSalt(){  
 this.bread.salt = new Salt(5);  
 }  
}

Prototype

package prac6;  
  
public class Prototype{  
 public Prototype(){  
 Work();  
 }  
  
 private void Work(){  
 var figure1 = new Rectangle(20, 40);  
 var figure1\_1 = figure1.Clone();  
 figure1.GetInfo();  
 figure1\_1.GetInfo();  
  
 var figure2 = new Circle(5);  
 var figure2\_1 = figure2.Clone();  
 figure2.GetInfo();  
 figure2\_1.GetInfo();  
 }  
}  
  
interface IFigure{  
 IFigure Clone();  
 void GetInfo();  
}  
  
class Rectangle implements IFigure{  
 private int X;  
 private int Y;  
  
 public Rectangle(int x, int y){  
 X = x;  
 Y = y;  
 }  
  
 public IFigure Clone(){  
 return new Rectangle(this.X, this.Y);  
 }  
  
 public void GetInfo(){  
 System.*out*.println("Прямоугольник: X = " + X + ", Y = " + Y);  
 }  
}  
  
class Circle implements IFigure {  
 private int Radius;  
  
 public Circle(int radius) {  
 Radius = radius;  
 }  
  
 public IFigure Clone() {  
 return new Circle(this.Radius);  
 }  
  
 public void GetInfo() {  
 System.*out*.println("Круг: Радиус = " + Radius);  
 }  
}

1. **Результат работы программы**



**Практическая работа №7**

1. **Постановка задачи и персональный вариант**

Цель: Реализация структурных паттернов проектирования.

Постановка задачи: написать реализацию паттернов Адаптер и Фасад.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac7>

Adapter.java

package prac7;  
  
public class Adapter{  
 public Adapter(){  
 Driver driver = new Driver();  
 vehicle vehicle = new vehicle();  
 driver.Travel(vehicle);  
  
 Camel camel = new Camel();  
 ITransport camelTransport = new CamelToTransportAdapter(camel);  
 driver.Travel(camelTransport);  
 }  
}  
  
class Driver{  
 public void Travel(ITransport transport){  
 transport.Drive();  
 }  
}  
  
interface ITransport{  
 void Drive();  
}  
class vehicle implements ITransport{  
 public void Drive(){  
 System.*out*.println("vehicle drived");  
 }  
}  
  
interface IAnimal{  
 void Move();  
}  
class Camel implements IAnimal{  
 public void Move(){  
 System.*out*.println("camel moved");  
 }  
}  
  
class CamelToTransportAdapter implements ITransport{  
 Camel camel;  
 public CamelToTransportAdapter(Camel c){  
 camel = c;  
 }  
  
 public void Drive(){  
 camel.Move();  
 }  
}

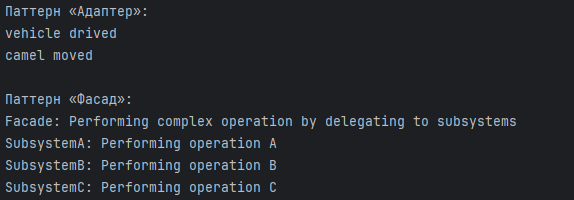
Facade.java

package prac7;  
  
class SubsystemA {  
 public void operationA() {  
 System.*out*.println("SubsystemA: Performing operation A");  
 }  
}  
  
class SubsystemB {  
 public void operationB() {  
 System.*out*.println("SubsystemB: Performing operation B");  
 }  
}  
  
class SubsystemC {  
 public void operationC() {  
 System.*out*.println("SubsystemC: Performing operation C");  
 }  
}  
public class Facade {  
 private SubsystemA subsystemA;  
 private SubsystemB subsystemB;  
 private SubsystemC subsystemC;  
  
 public Facade() {  
 this.subsystemA = new SubsystemA();  
 this.subsystemB = new SubsystemB();  
 this.subsystemC = new SubsystemC();  
 }  
  
 public void performComplexOperation() {  
 System.*out*.println("Facade: Performing complex operation by delegating to subsystems");  
 subsystemA.operationA();  
 subsystemB.operationB();  
 subsystemC.operationC();  
 }  
}

Main.java

package prac7;  
  
class SubsystemA {  
 public void operationA() {  
 System.*out*.println("SubsystemA: Performing operation A");  
 }  
}  
  
class SubsystemB {  
 public void operationB() {  
 System.*out*.println("SubsystemB: Performing operation B");  
 }  
}  
  
class SubsystemC {  
 public void operationC() {  
 System.*out*.println("SubsystemC: Performing operation C");  
 }  
}  
public class Facade {  
 private SubsystemA subsystemA;  
 private SubsystemB subsystemB;  
 private SubsystemC subsystemC;  
  
 public Facade() {  
 this.subsystemA = new SubsystemA();  
 this.subsystemB = new SubsystemB();  
 this.subsystemC = new SubsystemC();  
 }  
  
 public void performComplexOperation() {  
 System.*out*.println("Facade: Performing complex operation by delegating to subsystems");  
 subsystemA.operationA();  
 subsystemB.operationB();  
 subsystemC.operationC();  
 }  
}

1. **Результат работы программы**



**Практическая работа №8**

1. **Постановка задачи и персональный вариант**

Цель: Реализация поведенческих паттернов проектирования.

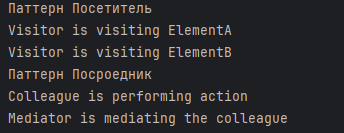
Постановка задачи: написать реализацию паттернов Посетитель и Посредник.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/practice/src/prac8>

package prac8;  
  
import java.util.ArrayList;  
import java.util.List;  
  
//ПОСЕТИТЕЛЬ  
interface Visitor {  
 void visit(ElementA element);  
 void visit(ElementB element);  
}  
  
class ConcreteVisitor implements Visitor {  
 @Override  
 public void visit(ElementA element) {  
 System.*out*.println("Visitor is visiting ElementA");  
 }  
  
 @Override  
 public void visit(ElementB element) {  
 System.*out*.println("Visitor is visiting ElementB");  
 }  
}  
  
interface Element {  
 void accept(Visitor visitor);  
}  
  
class ElementA implements Element {  
 @Override  
 public void accept(Visitor visitor) {  
 visitor.visit(this);  
 }  
}  
  
class ElementB implements Element {  
 @Override  
 public void accept(Visitor visitor) {  
 visitor.visit(this);  
 }  
}  
  
  
//ПОСРЕДНИК  
interface Mediator {  
 void mediate(Colleague colleague);  
}  
  
class ConcreteMediator implements Mediator {  
 @Override  
 public void mediate(Colleague colleague) {  
 System.*out*.println("Mediator is mediating the colleague");  
 // Perform mediation logic here  
 }  
}  
  
interface Colleague {  
 void action();  
}  
  
class ConcreteColleague implements Colleague {  
 private Mediator mediator;  
  
 public ConcreteColleague(Mediator mediator) {  
 this.mediator = mediator;  
 }  
  
 @Override  
 public void action() {  
 System.*out*.println("Colleague is performing action");  
 // Perform some action  
 mediator.mediate(this);  
 }  
}  
  
// Main class  
public class Main {  
 public static void main(String[] args) {  
 System.*out*.println("Паттерн Посетитель");  
 List<Element> elements = new ArrayList<>();  
 elements.add(new ElementA());  
 elements.add(new ElementB());  
  
 Visitor visitor = new ConcreteVisitor();  
 for (Element element : elements) {  
 element.accept(visitor);  
 }  
  
 System.*out*.println("Паттерн Посроедник");  
 Mediator mediator = new ConcreteMediator();  
 Colleague colleague = new ConcreteColleague(mediator);  
 colleague.action();  
 }  
}

1. **Результат работы программы**



**Практическая работа №9**

1. **Постановка задачи и персональный вариант**

Цель: Знакомство с системой сборки приложения. Gradle.

Постановка задачи: создать приложение, которое выводит какое-то сообщение в консоль. Создать Gradle Task, который создает jar-файл приложения, переносит его в отдельную папку, в которой хранится Dockerfile для jar, а затем создает Docker контейнер из данного jar-файла и запускает его.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task9>

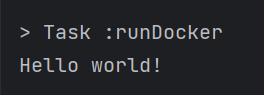
Класс Main:

public class Main {  
 public static void main(String[] args) {  
 System.*out*.println("Hello world!");  
 }  
}

Файл: build.gradle:

plugins {  
 id("java")  
}  
  
group = "org.example"  
version = "1.0-SNAPSHOT"  
  
repositories {  
 mavenCentral()  
}  
  
dependencies {  
 testImplementation 'org.junit.jupiter:junit-jupiter-api:5.7.2'  
 testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.7.2'  
}  
  
jar {  
 manifest {  
 attributes 'Main-Class': 'Main'  
 }  
}  
task buildDocker(type: Exec) {  
 dependsOn jar  
  
 workingDir '.'  
  
 commandLine 'docker', 'build', '-t', 'test', '.'  
}  
  
task runDocker(type: Exec) {  
 dependsOn buildDocker  
  
 workingDir '.'  
  
 commandLine 'docker', 'run', '--rm', 'test'  
}

1. **Результат работы программы**



**Практическая работа №10**

1. **Постановка задачи и персональный вариант**

Цель: Введение в Spring. Container. Bean. Внедрение зависимостей, основанных на конструкторах и сеттерах. Конфигурация бинов. Автоматическое обнаружение и связывание классов.

Постановка задачи: создать приложение, в котором создается ApplicationContext и из него берётся бин с названием, переданным в качестве аргумента к приложению, и вызывается метод интерфейса, который он имплементирует. Нужно создать по одному бину для каждого класса, определить им название. Проверить, что вызывается при вводе названия каждого из бинов. Классы и интерфейс: Интерфейс Programmer с методом doCoding(), его имплементации: Junior, Middle, Senior.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task10>

Класс Task10Application:

import config.BeanConfig;  
import interfaces.Programmer;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.AnnotationConfigApplicationContext;  
  
public class Task10Application {  
 public static void main(String[] args)  
 {  
 ApplicationContext context = new AnnotationConfigApplicationContext(BeanConfig.class);  
  
 Programmer bean = context.getBean("GetJunior", Programmer.class);  
 bean.doCoding();  
 bean = context.getBean("GetMiddle", Programmer.class);  
 bean.doCoding();  
 bean = context.getBean("GetSenior", Programmer.class);  
 bean.doCoding();  
 }  
}

Класс BeanConfig:

package config;  
  
import component.Junior;  
import component.Middle;  
import component.Senior;  
import interfaces.Programmer;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.ComponentScan;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
@ComponentScan  
public class BeanConfig {  
 @Bean  
 public Programmer GetJunior(){  
 return new Junior();  
 }  
 @Bean  
 public Programmer GetMiddle(){  
 return new Middle();  
 }  
 @Bean  
 public Programmer GetSenior(){  
 return new Senior();  
 }  
}

Класс Junior:

package component;  
  
import interfaces.Programmer;  
import org.springframework.stereotype.Component;  
  
@Component  
public class Junior implements Programmer {  
 @Override  
 public void doCoding() {  
 System.*out*.println("Junior coding");  
 }  
}

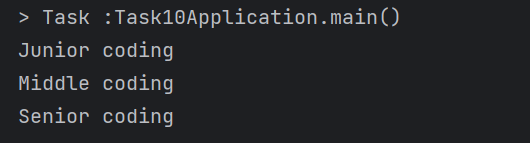
Класс Middle:

package component;  
import interfaces.Programmer;  
import org.springframework.stereotype.Component;  
  
@Component  
public class Middle implements Programmer {  
 @Override  
 public void doCoding() {  
 System.*out*.println("Middle coding");  
 }  
}

Класс Senior:

package component;  
import interfaces.Programmer;  
import org.springframework.stereotype.Component;  
  
@Component  
public class Senior implements Programmer {  
 @Override  
 public void doCoding() {  
 System.*out*.println("Senior coding");  
 }  
}

1. **Результат работы программы**



**Практическая работа №11**

1. **Постановка задачи и персональный вариант**

Цель: разобраться с использованием Spring boot.

Постановка задачи: создать приложение с использованием Spring Boot Starter Initializr (https://start.spring.io/) с такими зависимостями:

– Spring Web;

– Lombok;

– Validation;

– Spring boot Actuator.

Запустить приложение и удостовериться, что не появилось никаких ошибок. Добавить все эндпоинты в Actuator, сделать HTTP-запрос на проверку состояния приложения. Собрать jar-файл приложения, запустить и проверить состояние при помощи REST-запроса.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task11>

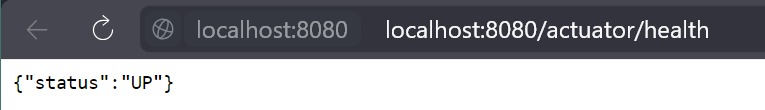
Класс Task11Application:

package com.example.task11;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class Task11Application {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(Task11Application.class, args);  
 }  
  
}

Класс HealthCheckController:

package com.example.task11.controllers;  
  
import org.springframework.web.bind.annotation.GetMapping;  
  
public class HealthCheckController {  
 @GetMapping("/health")  
 public String checkHealth() {  
 return "Application is up and running!";  
 }  
}

1. **Результат работы программы**



**Практическая работа №12**

1. **Постановка задачи и персональный вариант**

Цель: Работа с жизненным циклом компонентов. Аннотации PostConstruct, PreDestroy.

Постановка задачи: cоздать приложение, которое при запуске берет данные из одного файла, хеширует, а при остановке приложения удаляет исходный файл, оставляя только файл с захешированными данными. Названия первого и второго файла передаются в качестве аргументов при запуске. При отсутствии первого файла создает второй файл и записывает в него строку null. Реализовать с использованием аннотаций PostConstruct, PreDestroy.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task12>

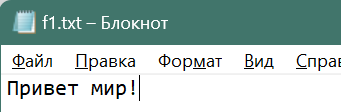
Класс Task12Application:

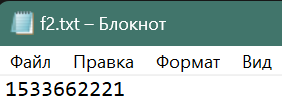
package com.example.task11;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class Task11Application {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(Task11Application.class, args);  
 }  
  
}

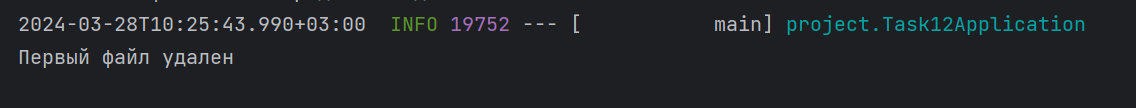
Класс EditFiles:

package org;  
  
import jakarta.annotation.PostConstruct;  
import jakarta.annotation.PreDestroy;  
import org.springframework.stereotype.Service;  
  
import java.io.\*;  
  
@Service  
public class EditFiles  
{  
 String file1 = "D:\\code\\\_4\\java\\task12\\f1.txt";  
 String file2 = "D:\\code\\\_4\\java\\task12\\f2.txt";  
  
 @PostConstruct  
 public void PostConstruct() throws IOException{  
 var bufferedWriter = new BufferedWriter(new FileWriter(file2));  
  
 if (new File(file1).exists()){  
 var bufferedReader = new BufferedReader(new FileReader(file1));  
 var strBuilder = new StringBuilder();  
 String line;  
  
 while ((line = bufferedReader.readLine()) != null){  
 strBuilder.append(line);  
 }  
 bufferedWriter.write(String.*valueOf*(strBuilder.hashCode()));  
 System.*out*.println("Текст из 1 файла был передан в виде хеша во 2");  
 bufferedReader.close();  
 }  
 else {  
 System.*out*.println("Первого файла нет");  
 bufferedWriter.write("null");  
 }  
 bufferedWriter.close();  
 }  
  
 @PreDestroy  
 public void PreDestroy(){  
 if(new File(file1).delete()){  
 System.*out*.println("Первый файл удален");  
 }  
 }  
}

1. **Результат работы программы**







# **Практическая работа №13**

1. **Постановка задачи и персональный вариант**

Тема: Конфигурирование приложения. Environment.

Постановка задачи: создать файл application.yml в папке resources, добавить в него такие свойства:

– student.name – имя студента;

– student.last\_name – фамилия студента;

– student.group – название группы студента.

При запуске приложения выведите данные свойства в консоль при помощи интерфейса Environment или аннотации Value.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task13>

Файл: Task13Application.java

package org.task13;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class Task13Application {  
 public static void main(String[] args) {  
 SpringApplication.*run*(Task13Application.class, args);  
 new User();  
 }  
}

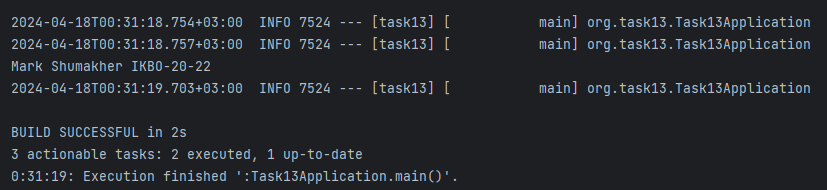
Файл: User.java

package org.task13;  
  
import jakarta.annotation.PostConstruct;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
@Component  
public class User {  
 @Value("${student.name}")  
 private String firstName;  
 @Value("${student.last\_name}")  
 private String lastName;  
 @Value("${student.group}")  
 private String group;  
  
 @PostConstruct  
 public void print() {  
 System.out.println(firstName + " " + lastName + " " + group);  
 }  
}

Файл: application.properties

spring.application.name=task13  
student.name=Mark  
student.last\_name=Shumakher  
student.group=IKBO-20-22

1. **Результат работы программы**



# **Практическая работа №14**

1. **Постановка задачи и персональный вариант**

Тема: Знакомство со Spring MVC. Работа с Rest API в Spring.

Постановка задачи: создать класс Footballer с полями firstName, lastName. Создать класс Team с полями name, creationDate. Создать классы-контроллеры для создания, удаления объектов и получения всех объектов каждого типа. Сами объекты хранить в памяти.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task14>

Файл: FootballerController.java

package org.task14.controller;  
  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.task14.entity.Footballer;  
  
import java.util.ArrayList;  
import java.util.List;  
  
@RestController  
@RequestMapping("footballer")  
public class FootballerController {  
 private List<Footballer> footballers = new ArrayList<>();  
  
 @PostMapping("/add")  
 public ResponseEntity AddFb(@RequestBody Footballer footballer)  
 {  
 footballers.add(footballer);  
 return ResponseEntity.*ok*(HttpStatus.*OK*);  
 }  
 // Delete  
 @DeleteMapping("/delete/{id}")  
 public ResponseEntity DeleteFb(@PathVariable int id)  
 {  
 footballers.remove(id);  
 return ResponseEntity.*ok*(HttpStatus.*OK*);  
 }  
 // GetAll  
 @GetMapping()  
 public List<Footballer> GetAllFb()  
 {  
 return footballers;  
 }  
}

Файл: TeamController.java

package org.task14.controller;  
  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.task14.entity.Team;  
  
import java.util.ArrayList;  
import java.util.List;  
  
@RestController  
@RequestMapping("team")  
public class TeamController {  
 private List<Team> teams = new ArrayList<>();  
  
 @PostMapping("/add")  
 public ResponseEntity AddTeam(@RequestBody Team team)  
 {  
 teams.add(team);  
 return ResponseEntity.*ok*(HttpStatus.*OK*);  
 }  
 // Delete  
 @DeleteMapping("/delete/{id}")  
 public ResponseEntity DeleteTeam(@PathVariable int id)  
 {  
 teams.remove(id);  
 return ResponseEntity.*ok*(HttpStatus.*OK*);  
 }  
 // GetAll  
 @GetMapping()  
 public List<Team> GetAllTeam()  
 {  
 return teams;  
 }  
}

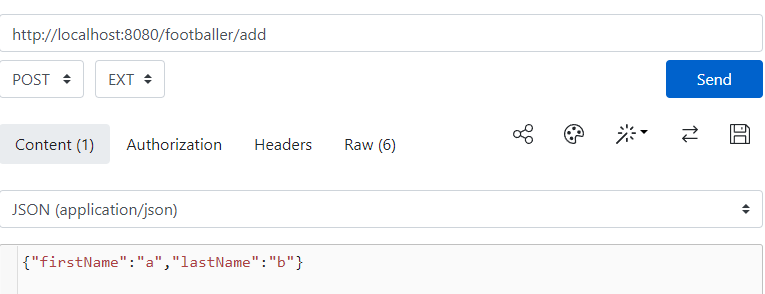
Файл: footballer.java

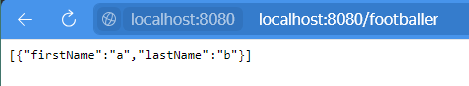
package org.task14.entity;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.Setter;  
  
@Getter  
@Setter  
@AllArgsConstructor  
public class Footballer {  
 private String firstName;  
 private String lastName;  
}

Файл: Team.java

package org.task14.entity;  
  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.Setter;  
  
@Getter  
@Setter  
@AllArgsConstructor  
public class Team {  
 private String name;  
 private String creationData;  
}

1. **Результат работы программы**





# **Практическая работа №15**

1. **Постановка задачи и персональный вариант**

Тема: Использование Hibernate в Spring framework.

Постановка задачи: изменить программу с предыдущего задания так, чтобы объекты хранились в базе данных PostgreSQL вместо памяти компьютера.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task15>

Файл: Team.java

package org.task15.entity;  
  
import jakarta.persistence.\*;  
import lombok.Getter;  
import lombok.Setter;  
  
@Entity  
@Table(name = "teams")  
@Getter  
@Setter  
public class Team  
{  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 @Column(name="name", nullable=false, length=25)  
 private String name;  
 @Column(name="creation\_date", nullable=false, length=25)  
 private String creationDate;  
}

Файл: TeamController.java

package org.task15.controller;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.task15.service.TeamService;  
  
import java.sql.SQLException;  
import java.util.List;  
  
@RestController  
@RequestMapping("api/teams")  
public class TeamController  
{  
 @Autowired  
 private TeamService teamService;  
  
 @PostMapping()  
 public ResponseEntity AddTeam(@RequestBody org.task15.entity.Team team) throws SQLException  
 {  
 teamService.AddTeam(team);  
 return ResponseEntity.ok(HttpStatus.OK);  
 }  
  
 @DeleteMapping("/{id}")  
 public ResponseEntity DeleteTeam(@PathVariable Long id) throws SQLException  
 {  
 teamService.DeleteTeam(id);  
 return ResponseEntity.ok(HttpStatus.OK);  
 }  
  
 @GetMapping()  
 public List<org.task15.entity.Team> GetAllTeam() throws SQLException  
 {  
 return teamService.GetTeams();  
 }  
  
 @GetMapping("/{id}")  
 public org.task15.entity.Team GetTeamById(@PathVariable Long id) throws SQLException  
 {  
 return teamService.GetTeamById(id);  
 }  
}

Файл: TeamService.java

package org.task15.service;  
  
import lombok.AllArgsConstructor;  
import org.springframework.stereotype.Service;  
import org.task15.entity.Team;  
import org.task15.repository.TeamRepository;  
  
import java.util.List;  
  
@Service  
@AllArgsConstructor  
public class TeamService  
{  
 private TeamRepository teamRepository;  
  
 public void AddTeam(Team team)  
 {  
 teamRepository.AddTeam(team);  
 }  
  
 public void DeleteTeam(Long id)  
 {  
 teamRepository.DeleteTeam(id);  
 }  
  
 public List<Team> GetTeams()  
 {  
 return teamRepository.GetAllTeam();  
 }  
  
 public Team GetTeamById(Long id)  
 {  
 return teamRepository.GetTeamById(id);  
 }  
}

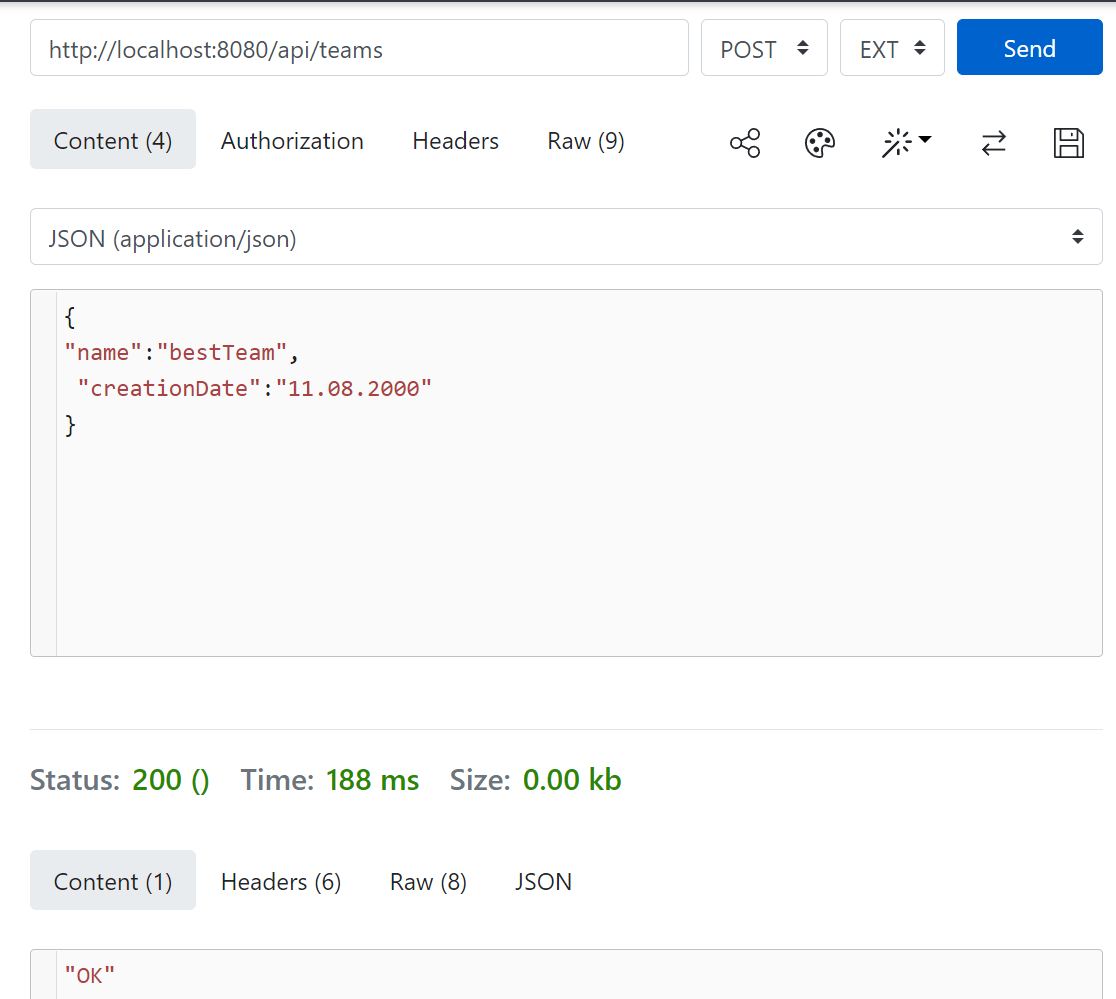
Файл: TeamRepository.java

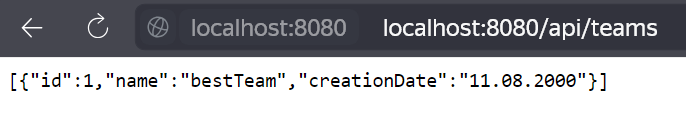
package org.task15.repository;  
  
import jakarta.annotation.PostConstruct;  
import lombok.RequiredArgsConstructor;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.springframework.stereotype.Repository;  
import org.task15.entity.Team;  
  
import java.util.List;  
  
@Repository  
@RequiredArgsConstructor  
public class TeamRepository  
{  
 private final SessionFactory sessionFactory;  
 private Session session;  
  
 @PostConstruct  
 void init()  
 {  
 session = sessionFactory.openSession();  
 }  
  
 public void AddTeam(Team team)  
 {  
 var t = session.beginTransaction();  
 session.persist(team);  
 t.commit();  
 }  
  
 public void DeleteTeam(Long id)  
 {  
 var t = session.beginTransaction();  
 session.remove(session.find(Team.class, id));  
 t.commit();  
 }  
  
 public List<Team> GetAllTeam()  
 {  
 var t = session.beginTransaction();  
 var teams = session.createQuery("select g from Team g", Team.class).getResultList();  
 t.commit();  
 return teams;  
 }  
  
 public Team GetTeamById(Long id)  
 {  
 var t = session.beginTransaction();  
 var team = session.find(Team.class, id);  
 t.commit();  
 return team;  
 }  
}

Файл: BeanConfig.java

package org.task15;  
  
import com.zaxxer.hikari.HikariConfig;  
import com.zaxxer.hikari.HikariDataSource;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.orm.hibernate5.HibernateTransactionManager;  
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;  
import org.springframework.transaction.PlatformTransactionManager;  
  
import javax.sql.DataSource;  
import java.util.Properties;  
  
@Configuration  
public class BeanConfig  
{  
 @Bean  
 public HikariDataSource dataSource()  
 {  
 var config = new HikariConfig();  
 config.setJdbcUrl("jdbc:postgresql://localhost:5432/postgres");  
 config.setUsername("postgres");  
 config.setPassword("password");  
  
 return new HikariDataSource(config);  
 }  
  
 @Bean(name="entityManagerFactory")  
 public LocalSessionFactoryBean factoryBean(DataSource dataSource)  
 {  
 var sessionFactoryBean = new LocalSessionFactoryBean();  
 sessionFactoryBean.setDataSource(dataSource);  
 sessionFactoryBean.setPackagesToScan("org.task15");  
  
 var properties = new Properties();  
 properties.setProperty("hibernate.dialect", "org.hibernate.dialect.PostgreSQLDialect");  
 properties.setProperty("hibernate.hbm2ddl.auto", "create");  
 sessionFactoryBean.setHibernateProperties(properties);  
  
 return sessionFactoryBean;  
 }  
  
 @Bean  
 public PlatformTransactionManager platformTransactionManager(LocalSessionFactoryBean factoryBean)  
 {  
 var transactionManager = new HibernateTransactionManager();  
 transactionManager.setSessionFactory(factoryBean.getObject());  
 return transactionManager;  
 }  
}

1. **Результат работы программы**





# **Практическая работа №16**

1. **Постановка задачи и персональный вариант**

Тема: Изучение видов связей между сущностями в Hibernate. Использование транзакций.

Постановка задачи: создать связь Один-ко-многим между сущностями из предыдущего задания и проверить работу lazy loading.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task16>

Файл: Team.java

package org.task16.entity;  
  
import jakarta.persistence.\*;  
import lombok.Getter;  
import lombok.Setter;  
  
import java.util.List;  
  
@Entity  
@Table(name = "teams")  
@Getter  
@Setter  
public class Team  
{  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name="name", length=25)  
 private String name;  
 @Column(name="creation\_date", length=25)  
 private String creationDate;  
 @OneToMany(cascade = CascadeType.*ALL*)  
 @JoinColumn(name = "team\_id")  
 private List<Footballer> footballers;  
}

Файл: Footballer.java

package org.task16.entity;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.Setter;  
  
@Entity  
@Getter  
@Setter  
@Table(name="footballers")  
public class Footballer {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name="firstName", nullable=false, length=25)  
 private String firstName;  
 @Column(name="lastName", nullable=false, length=25)  
 private String lastName;  
  
  
  
 @Column(name="team\_id")  
 public Long teamId;  
}

Файл: TeamController.java

package org.task16.controller;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.task16.entity.Team;  
import org.task16.service.TeamService;  
  
import java.sql.SQLException;  
import java.util.List;  
  
@RestController  
@RequestMapping("api/teams")  
public class TeamController  
{  
 @Autowired  
 private TeamService teamService;  
  
 @PostMapping()  
 public ResponseEntity AddTeam(@RequestBody Team team) throws SQLException  
 {  
 teamService.AddTeam(team);  
 return ResponseEntity.*ok*(HttpStatus.*OK*);  
 }  
  
 @DeleteMapping("/{id}")  
 public ResponseEntity DeleteTeam(@PathVariable Long id) throws SQLException  
 {  
 teamService.DeleteTeam(id);  
 return ResponseEntity.*ok*(HttpStatus.*OK*);  
 }  
  
 @GetMapping()  
 public List<Team> GetAllTeam() throws SQLException  
 {  
 return teamService.GetTeams();  
 }  
  
 @GetMapping("/{id}")  
 public Team GetTeamById(@PathVariable Long id) throws SQLException  
 {  
 return teamService.GetTeamById(id);  
 }  
}

Файл: FootballerController.java

package org.task16.controller;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.task16.entity.Footballer;  
import org.task16.entity.Team;  
import org.task16.entity.Footballer;  
import org.task16.service.FootballerService;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("api/footballers")  
public class FootballerController  
{  
 @Autowired  
 private FootballerService footballerService;  
  
 @PostMapping("/{teamId}")  
 public ResponseEntity AddFootballer(@RequestBody Footballer level, @PathVariable Long teamId)  
 {  
 footballerService.AddFootballer(level, teamId);  
 return ResponseEntity.*ok*(HttpStatus.*OK*);  
 }  
  
 @DeleteMapping("/{id}")  
 public ResponseEntity DeleteFootballer(@PathVariable Long id)  
 {  
 footballerService.DeleteFootballer(id);  
 return ResponseEntity.*ok*(HttpStatus.*OK*);  
 }  
  
 @GetMapping()  
 public List<Footballer> GetAllFootballer()  
 {  
 return footballerService.GetAllFootballer();  
 }  
  
 @GetMapping("/{id}")  
 public Footballer GetFootballerById(@PathVariable Long id)  
 {  
 return footballerService.GetFootballerById(id);  
 }  
}

Файл: TeamService.java

package org.task16.service;  
  
import lombok.AllArgsConstructor;  
import org.springframework.stereotype.Service;  
import org.task16.entity.Team;  
import org.task16.repository.TeamRepository;  
  
import java.util.List;  
  
@Service  
@AllArgsConstructor  
public class TeamService  
{  
 private TeamRepository teamRepository;  
  
 public void AddTeam(Team team)  
 {  
 teamRepository.AddTeam(team);  
 }  
  
 public void DeleteTeam(Long id)  
 {  
 teamRepository.DeleteTeam(id);  
 }  
  
 public List<Team> GetTeams()  
 {  
 return teamRepository.GetAllTeam();  
 }  
  
 public Team GetTeamById(Long id)  
 {  
 return teamRepository.GetTeamById(id);  
 }  
}

Файл: FootballerService.java

package org.task16.service;  
  
import lombok.AllArgsConstructor;  
import org.springframework.stereotype.Service;  
import org.task16.entity.Team;  
import org.task16.entity.Footballer;  
import org.task16.repository.TeamRepository;  
import org.task16.repository.FootballerRepository;  
  
import java.util.List;  
  
@Service  
@AllArgsConstructor  
public class FootballerService  
{  
 private FootballerRepository teamRepository;  
  
 public void AddFootballer(Footballer f, Long teamId)  
 {  
 teamRepository.AddFootballer(f, teamId);  
 }  
  
 public void DeleteFootballer(Long id)  
 {  
 teamRepository.DeleteFootballer(id);  
 }  
  
 public List<Footballer> GetAllFootballer()  
 {  
 return teamRepository.GetAllFootballer();  
 }  
  
 public Footballer GetFootballerById(Long id)  
 {  
 return teamRepository.GetFootballerById(id);  
 }  
}

Файл: TeamRepository.java

package org.task16.repository;  
  
import jakarta.annotation.PostConstruct;  
import lombok.RequiredArgsConstructor;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.springframework.stereotype.Repository;  
import org.task16.entity.Footballer;  
import org.task16.entity.Team;  
  
import java.util.List;  
  
@Repository  
@RequiredArgsConstructor  
public class TeamRepository  
{  
 private final SessionFactory sessionFactory;  
 private Session session;  
  
 @PostConstruct  
 void init()  
 {  
 session = sessionFactory.openSession();  
 }  
  
 public void AddTeam(Team team)  
 {  
 var t = session.beginTransaction();  
 session.persist(team);  
 t.commit();  
 }  
  
 public void DeleteTeam(Long id)  
 {  
 var t = session.beginTransaction();  
 session.remove(session.find(Team.class, id));  
 t.commit();  
 }  
  
 public List<Team> GetAllTeam()  
 {  
 var t = session.beginTransaction();  
 var teams = session.createQuery("select g from Team g", Team.class).getResultList();  
 for (Team g : teams)  
 {  
 var footballers = session.createQuery("select l from Footballer l where l.teamId=%s".formatted(g.getId()), Footballer.class).getResultList();  
 g.setFootballers(footballers);  
 }  
 t.commit();  
 return teams;  
 }  
  
 public Team GetTeamById(Long id)  
 {  
 var t = session.beginTransaction();  
 var team = session.find(Team.class, id);  
 t.commit();  
 return team;  
 }  
}

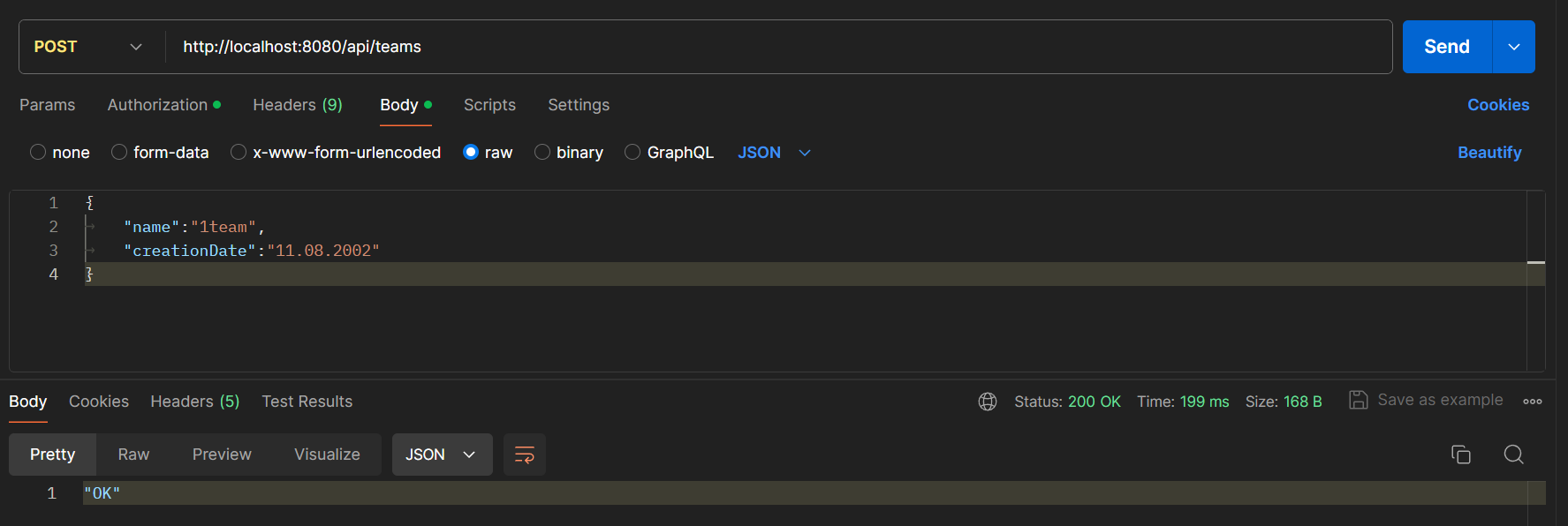
Файл: FootballerRepository.java

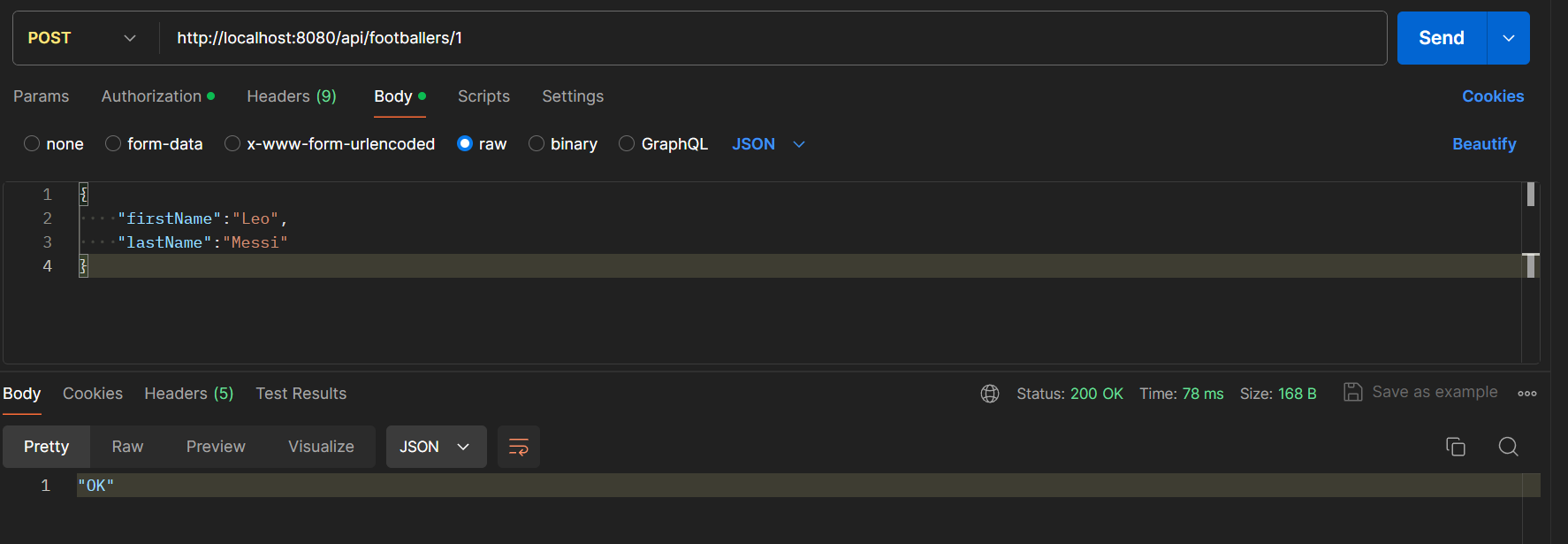
package org.task16.repository;  
  
import jakarta.annotation.PostConstruct;  
import lombok.RequiredArgsConstructor;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.springframework.stereotype.Repository;  
import org.task16.entity.Team;  
import org.task16.entity.Footballer;  
  
import java.util.List;  
  
@Repository  
@RequiredArgsConstructor  
public class FootballerRepository  
{  
 private final SessionFactory sessionFactory;  
 private Session session;  
  
 @PostConstruct  
 void init()  
 {  
 session = sessionFactory.openSession();  
 }  
  
 public void AddFootballer(Footballer footballer, Long teamId)  
 {  
 var t = session.beginTransaction();  
 var team = session.get(Team.class, teamId);  
 session.persist(footballer);  
  
 team.getFootballers().add(footballer);  
 session.update(team);  
 t.commit();  
  
 }  
  
 public void DeleteFootballer(Long id)  
 {  
 var t = session.beginTransaction();  
 session.remove(session.find(Footballer.class, id));  
 t.commit();  
 }  
  
 public List<Footballer> GetAllFootballer()  
 {  
 var t = session.beginTransaction();  
 var footballers = session.createQuery("select g from Footballer g", Footballer.class).getResultList();  
 t.commit();  
 return footballers;  
 }  
  
 public Footballer GetFootballerById(Long id)  
 {  
 var t = session.beginTransaction();  
 var team = session.find(Footballer.class, id);  
 t.commit();  
 return team;  
 }  
}

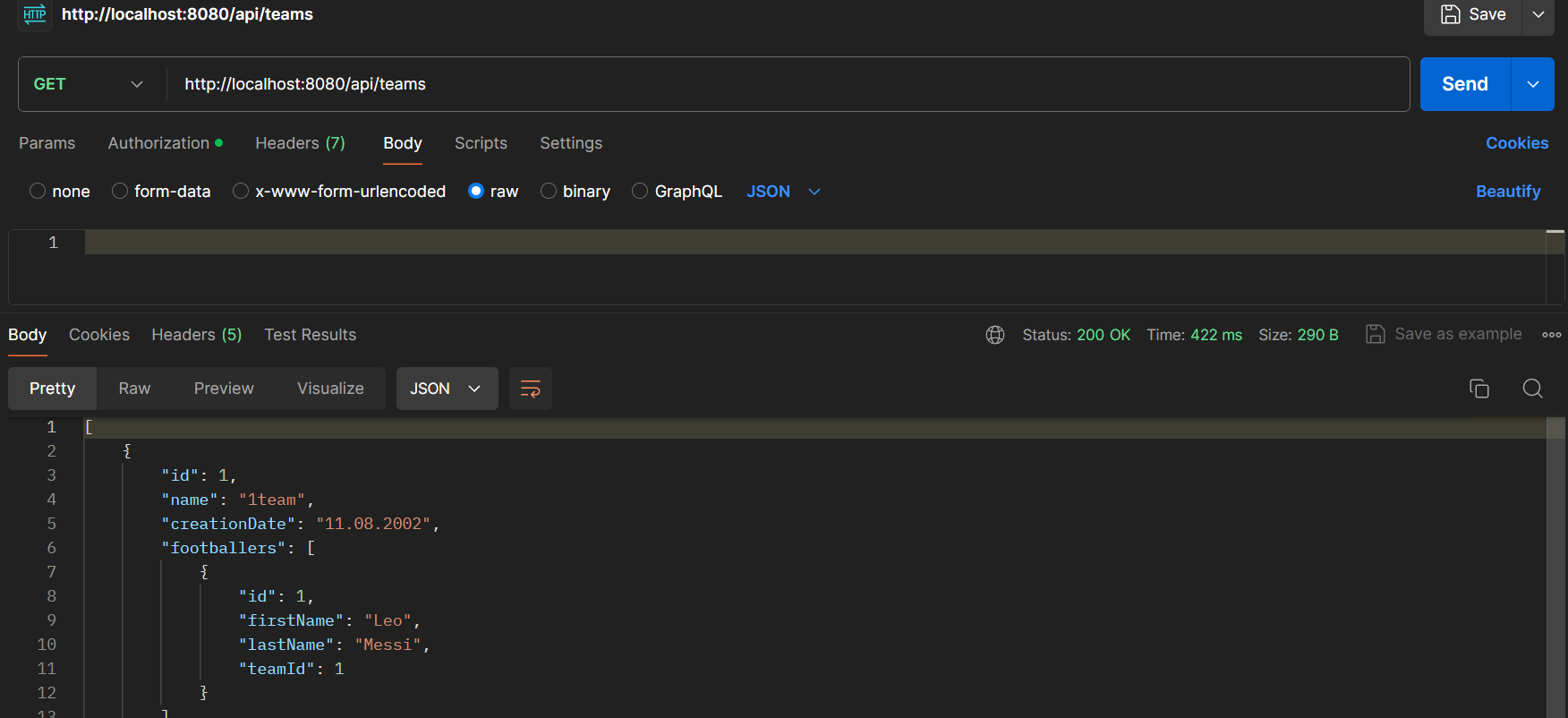
Файл: BeanConfig.java

package org.task16;  
  
import com.zaxxer.hikari.HikariConfig;  
import com.zaxxer.hikari.HikariDataSource;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.orm.hibernate5.HibernateTransactionManager;  
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;  
import org.springframework.transaction.PlatformTransactionManager;  
  
import javax.sql.DataSource;  
import java.util.Properties;  
  
@Configuration  
public class BeanConfig  
{  
 @Bean  
 public HikariDataSource dataSource()  
 {  
 var config = new HikariConfig();  
 config.setJdbcUrl("jdbc:postgresql://localhost:5432/test2");  
 config.setUsername("postgres");  
 config.setPassword("password");  
  
 return new HikariDataSource(config);  
 }  
  
 @Bean(name="entityManagerFactory")  
 public LocalSessionFactoryBean factoryBean(DataSource dataSource)  
 {  
 var sessionFactoryBean = new LocalSessionFactoryBean();  
 sessionFactoryBean.setDataSource(dataSource);  
 sessionFactoryBean.setPackagesToScan("org.task16");  
  
 var properties = new Properties();  
 properties.setProperty("hibernate.dialect", "org.hibernate.dialect.PostgreSQLDialect");  
 properties.setProperty("hibernate.hbm2ddl.auto", "create");  
 sessionFactoryBean.setHibernateProperties(properties);  
  
 return sessionFactoryBean;  
 }  
  
 @Bean  
 public PlatformTransactionManager platformTransactionManager(LocalSessionFactoryBean factoryBean)  
 {  
 var transactionManager = new HibernateTransactionManager();  
 transactionManager.setSessionFactory(factoryBean.getObject());  
 return transactionManager;  
 }  
}

1. **Результат работы программы**







# **Практическая работа №17**

1. **Постановка задачи и персональный вариант**

Тема: Знакомство с Criteria API в Hibernate.

Постановка задачи: добавить возможность фильтрации по всем полям всех классов с использованием Criteria API в Hibernate для программы из предыдущего задания. Добавить эндпоинты для каждой фильтрации.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task17>

Файл: Team.java

package org.task17.entity;  
  
import jakarta.persistence.\*;  
import lombok.Getter;  
import lombok.Setter;  
  
@Entity  
@Table(name = "teams")  
@Getter  
@Setter  
public class Team  
{  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 @Column(name="name", nullable=false, length=25)  
 private String name;  
 @Column(name="creation\_date", nullable=false, length=25)  
 private String creationDate;  
}

Файл: TeamController.java

package org.task17.controller;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.task17.entity.Team;  
import org.task17.service.TeamService;  
  
import java.sql.SQLException;  
import java.util.List;  
  
@RestController  
@RequestMapping("api/teams")  
public class TeamController  
{  
 @Autowired  
 private TeamService teamService;  
  
 @PostMapping()  
 public ResponseEntity AddTeam(@RequestBody org.task17.entity.Team team) throws SQLException  
 {  
 teamService.AddTeam(team);  
 return ResponseEntity.*ok*(HttpStatus.*OK*);  
 }  
  
 @DeleteMapping("/{id}")  
 public ResponseEntity DeleteTeam(@PathVariable Long id) throws SQLException  
 {  
 teamService.DeleteTeam(id);  
 return ResponseEntity.*ok*(HttpStatus.*OK*);  
 }  
  
 @GetMapping()  
 public List<org.task17.entity.Team> GetAllTeam() throws SQLException  
 {  
 return teamService.GetTeams();  
 }  
  
 @GetMapping("/{id}")  
 public org.task17.entity.Team GetTeamById(@PathVariable Long id) throws SQLException  
 {  
 return teamService.GetTeamById(id);  
 }  
 // Filter by name  
 @GetMapping("/name/{pattern}")  
 public List<Team> FilterTeamsByName(@PathVariable String pattern){  
 return teamService.FilterTeamsByName(pattern);  
 }  
  
 // Filter by creationDate  
 @GetMapping("/date/{pattern}")  
 public List<Team> FilterTeamsByDate(@PathVariable String pattern)  
 {  
 return teamService.FilterTeamsByDate(pattern);  
 }  
}

Файл: TeamService.java

package org.task17.service;  
  
import lombok.AllArgsConstructor;  
import org.springframework.stereotype.Service;  
import org.task17.entity.Team;  
import org.task17.repository.TeamRepository;  
  
import java.util.List;  
  
@Service  
@AllArgsConstructor  
public class TeamService  
{  
 private TeamRepository teamRepository;  
  
 public void AddTeam(Team team)  
 {  
 teamRepository.AddTeam(team);  
 }  
  
 public void DeleteTeam(Long id)  
 {  
 teamRepository.DeleteTeam(id);  
 }  
  
 public List<Team> GetTeams()  
 {  
 return teamRepository.GetAllTeam();  
 }  
  
 public Team GetTeamById(Long id)  
 {  
 return teamRepository.GetTeamById(id);  
 }  
  
 // Filter  
 public List<Team> FilterTeamsByName(String pattern)  
 {  
 return teamRepository.FilterTeamsByName(pattern);  
 }  
 public List<Team> FilterTeamsByDate(String pattern)  
 {  
 return teamRepository.FilterTeamsByDate(pattern);  
 }  
}

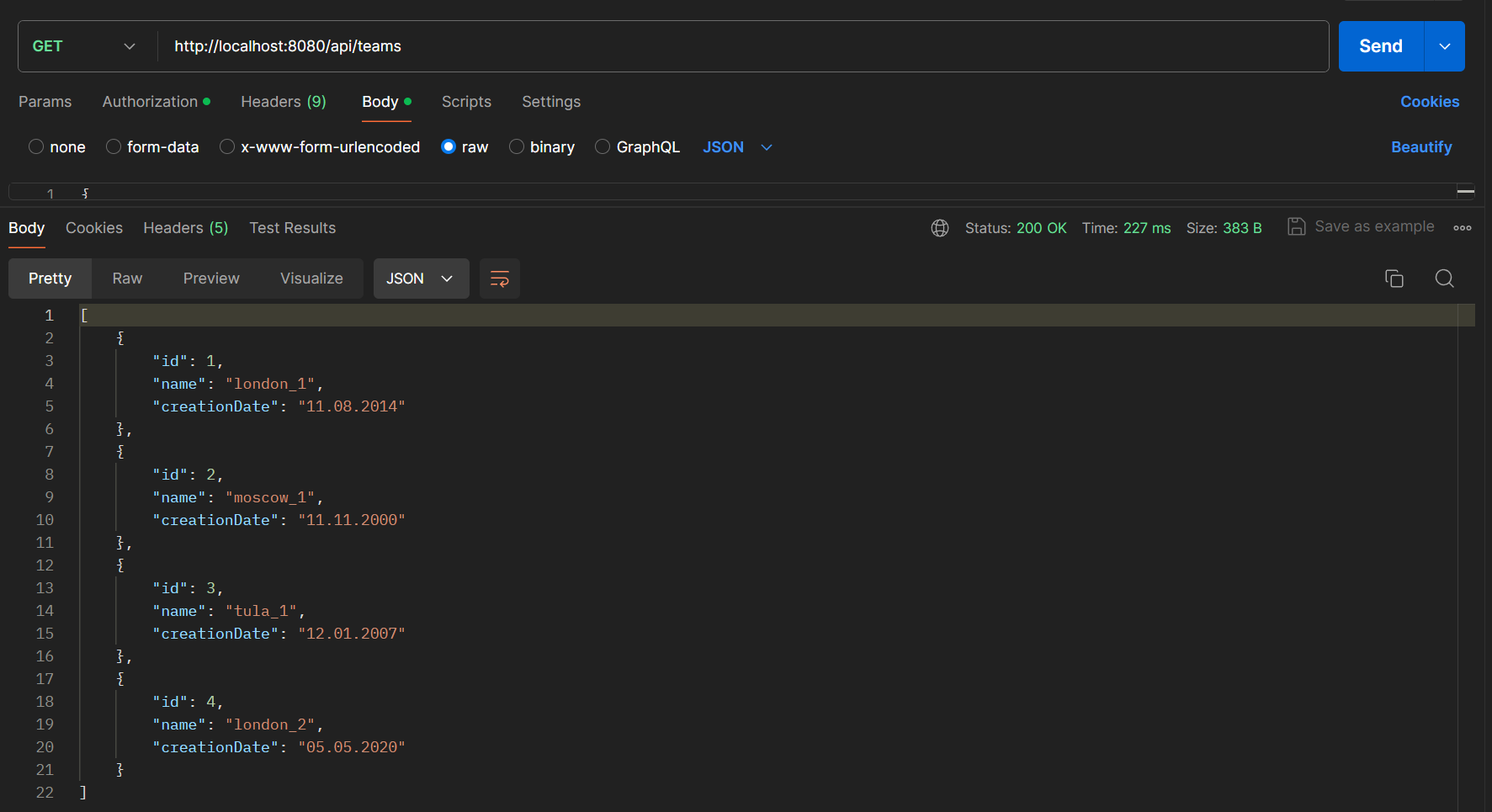
Файл: TeamRepository.java

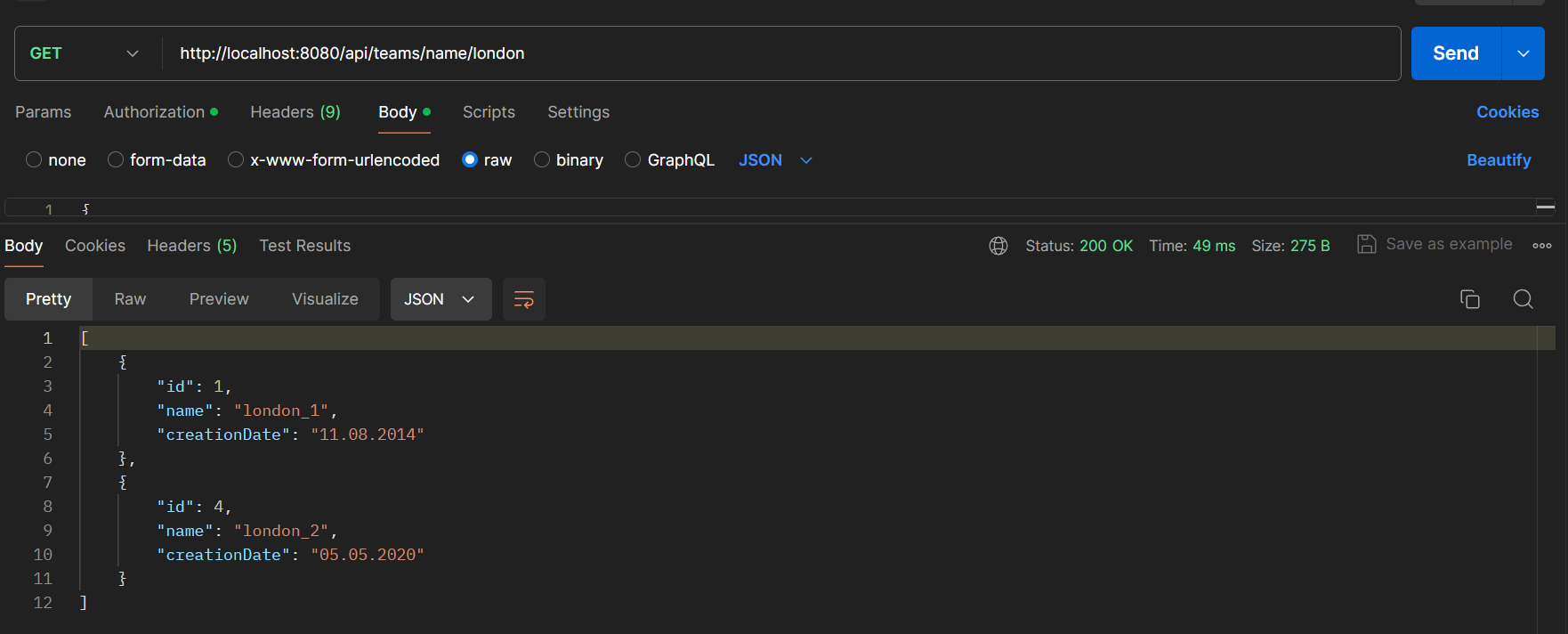
package org.task17.repository;  
  
import jakarta.annotation.PostConstruct;  
import jakarta.persistence.criteria.CriteriaBuilder;  
import jakarta.persistence.criteria.CriteriaQuery;  
import jakarta.persistence.criteria.Root;  
import lombok.RequiredArgsConstructor;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.springframework.stereotype.Repository;  
import org.task17.entity.Team;  
  
import java.util.List;  
  
@Repository  
@RequiredArgsConstructor  
public class TeamRepository  
{  
 private final SessionFactory sessionFactory;  
 private Session session;  
  
 @PostConstruct  
 void init()  
 {  
 session = sessionFactory.openSession();  
 }  
  
 public void AddTeam(Team team)  
 {  
 var t = session.beginTransaction();  
 session.persist(team);  
 t.commit();  
 }  
  
 public void DeleteTeam(Long id)  
 {  
 var t = session.beginTransaction();  
 session.remove(session.find(Team.class, id));  
 t.commit();  
 }  
  
 public List<Team> GetAllTeam()  
 {  
 var t = session.beginTransaction();  
 var teams = session.createQuery("select g from Team g", Team.class).getResultList();  
 t.commit();  
 return teams;  
 }  
  
 public Team GetTeamById(Long id)  
 {  
 var t = session.beginTransaction();  
 var team = session.find(Team.class, id);  
 t.commit();  
 return team;  
 }  
 // Filter  
 public List<Team> FilterTeamsByName(String pattern)  
 {  
 CriteriaBuilder builder = session.getCriteriaBuilder();  
 CriteriaQuery<Team> criteriaQuery = builder.createQuery(Team.class);  
 Root<Team> root = criteriaQuery.from(Team.class);  
  
 var pred = builder.like(root.get("name"), "%"+pattern+"%");  
 var query = criteriaQuery.select(root).where(pred);  
 return session.createQuery(query).getResultList();  
 }  
  
 public List<Team> FilterTeamsByDate(String pattern)  
 {  
 CriteriaBuilder builder = session.getCriteriaBuilder();  
 CriteriaQuery<Team> criteriaQuery = builder.createQuery(Team.class);  
 Root<Team> root = criteriaQuery.from(Team.class);  
  
 var pred = builder.like(root.get("creationDate"), "%"+pattern+"%");  
 var query = criteriaQuery.select(root).where(pred);  
 return session.createQuery(query).getResultList();  
 }  
}

Файл: BeanConfig.java

package org.task17;  
  
import com.zaxxer.hikari.HikariConfig;  
import com.zaxxer.hikari.HikariDataSource;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.orm.hibernate5.HibernateTransactionManager;  
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;  
import org.springframework.transaction.PlatformTransactionManager;  
  
import javax.sql.DataSource;  
import java.util.Properties;  
  
@Configuration  
public class BeanConfig  
{  
 @Bean  
 public HikariDataSource dataSource()  
 {  
 var config = new HikariConfig();  
 config.setJdbcUrl("jdbc:postgresql://localhost:5432/test1");  
 config.setUsername("postgres");  
 config.setPassword("password");  
  
 return new HikariDataSource(config);  
 }  
  
 @Bean(name="entityManagerFactory")  
 public LocalSessionFactoryBean factoryBean(DataSource dataSource)  
 {  
 var sessionFactoryBean = new LocalSessionFactoryBean();  
 sessionFactoryBean.setDataSource(dataSource);  
 sessionFactoryBean.setPackagesToScan("org.task17");  
  
 var properties = new Properties();  
 properties.setProperty("hibernate.dialect", "org.hibernate.dialect.PostgreSQLDialect");  
 properties.setProperty("hibernate.hbm2ddl.auto", "create");  
 sessionFactoryBean.setHibernateProperties(properties);  
  
 return sessionFactoryBean;  
 }  
  
 @Bean  
 public PlatformTransactionManager platformTransactionManager(LocalSessionFactoryBean factoryBean)  
 {  
 var transactionManager = new HibernateTransactionManager();  
 transactionManager.setSessionFactory(factoryBean.getObject());  
 return transactionManager;  
 }  
}

1. **Результат работы программы**





# **Практическая работа №18**

1. **Постановка задачи и персональный вариант**

Тема: Знакомство с репозиториями и сервисами, реализация в проекте. Взаимодействие с Spring Data JPA.

Постановка задачи: переписать код предыдущего задания с использованием сервисов и отделения логики контроллера от логики сервиса и репозитория. В программе всё взаимодействие с базой данных должно быть реализовано через репозитории Spring Data Jpa.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task18>

Файл: TeamRepository.java

package org.task18.repository;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
import org.task18.entity.Team;  
  
import java.util.List;  
  
@Repository  
public interface TeamRepository extends JpaRepository<Team, Long>  
{  
 List<Team> findAllByNameContains(String name);  
 List<Team> findAllByCreationDateContains(String date);  
}

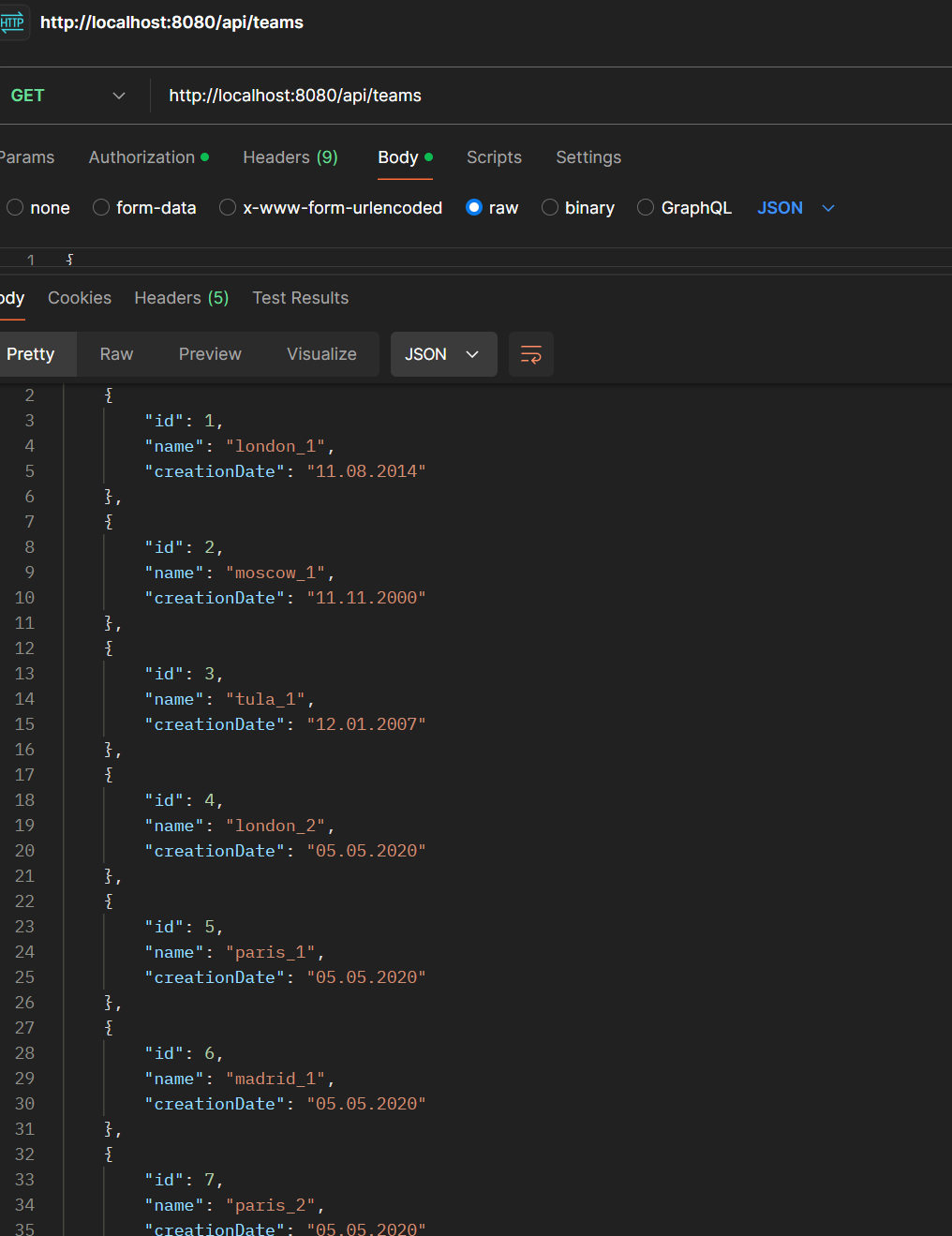
Файл: TeamService.java

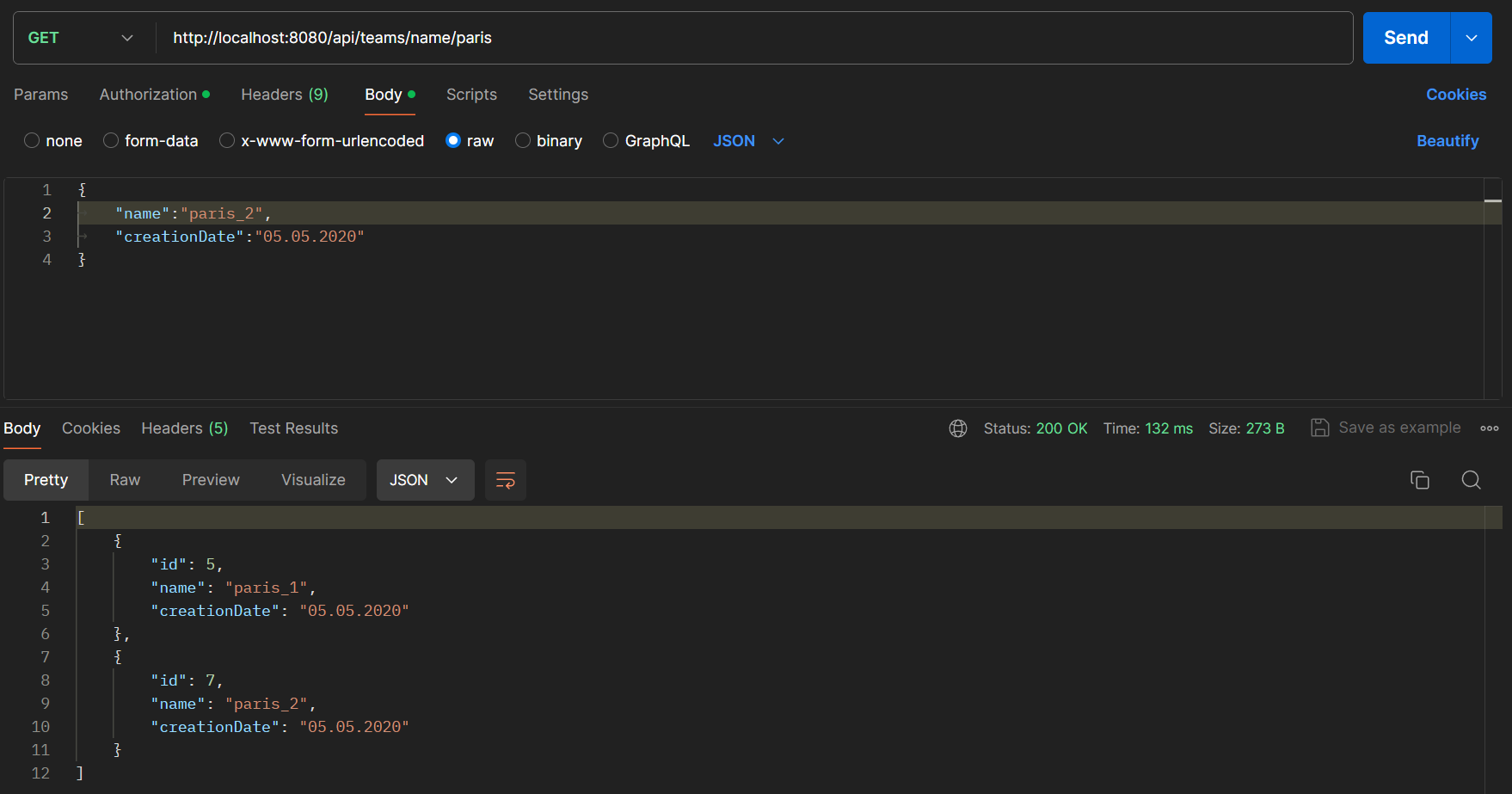
package org.task18.service;  
  
import lombok.AllArgsConstructor;  
import org.springframework.stereotype.Service;  
import org.task18.entity.Team;  
import org.task18.repository.TeamRepository;  
  
import java.util.List;  
  
@Service  
@AllArgsConstructor  
public class TeamService  
{  
 private TeamRepository teamRepository;  
  
 public void AddTeam(Team team)  
 {  
 teamRepository.save(team);  
 }  
  
 public void DeleteTeam(Long id)  
 {  
 teamRepository.deleteById(id);  
 }  
  
 public List<Team> GetTeams()  
 {  
 return teamRepository.findAll();  
 }  
  
 public Team GetTeamById(Long id)  
 {  
 return teamRepository.findById(id).get();  
 }  
  
 // Filter  
 public List<Team> FilterTeamsByName(String pattern)  
 {  
 return teamRepository.findAllByNameContains(pattern);  
 }  
 public List<Team> FilterTeamsByDate(String pattern)  
 {  
 return teamRepository.findAllByCreationDateContains(pattern);  
 }  
}

Файл: application.properties

spring.application.name=Task18  
spring.datasource.driver-class-name=org.postgresql.Driver  
spring.jpa.show-sql=true  
spring.jpa.generate-ddl=true  
spring.datasource.url=jdbc:postgresql://localhost:5432/test1  
spring.datasource.username=postgres  
spring.datasource.password=password

1. **Результат работы программы**





# **Практическая работа №19**

1. **Постановка задачи и персональный вариант**

Тема: Знакомство с логированием с использованием Logback в Spring.

Постановка задачи: создать файл logback.xml, добавить логирование во все методы классов-сервисов.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task19>

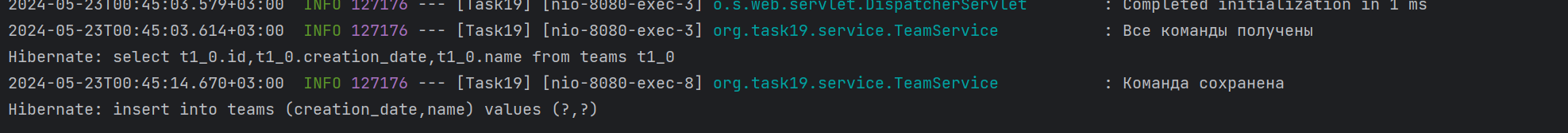
Файл: TeamService.java

package org.task19.service;  
  
import lombok.AllArgsConstructor;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.stereotype.Service;  
import org.task19.entity.Team;  
import org.task19.repository.TeamRepository;  
  
import java.util.List;  
  
@Service  
@AllArgsConstructor  
@Slf4j  
public class TeamService  
{  
 private TeamRepository teamRepository;  
  
 public void AddTeam(Team team)  
 {  
 *log*.info("Команда сохранена");  
 teamRepository.save(team);  
 }  
  
 public void DeleteTeam(Long id)  
 {  
 teamRepository.deleteById(id);  
 }  
  
 public List<Team> GetTeams()  
 {  
 *log*.info("Все команды получены");  
 return teamRepository.findAll();  
 }  
  
 public Team GetTeamById(Long id)  
 {  
 return teamRepository.findById(id).get();  
 }  
  
 // Filter  
 public List<Team> FilterTeamsByName(String pattern){  
 return teamRepository.findAllByNameContains(pattern);  
 }  
 public List<Team> FilterTeamsByDate(String pattern){  
 return teamRepository.findAllByCreationDateContains(pattern);  
 }  
}

Файл: logback.xml

<?xml version="1.0" encoding="UTF-8"?>  
<configuration>  
  
 <property name="LOGS" value="./logs" />  
  
 <appender name="Console"  
 class="ch.qos.logback.core.ConsoleAppender">  
 <layout class="ch.qos.logback.classic.PatternLayout">  
 <Pattern>  
 %black(%d{ISO8601}) %highlight(%-5level) [%blue(%t)] %yellow(%C{1.}): %msg%n%throwable  
 </Pattern>  
 </layout>  
 </appender>  
  
 <appender name="RollingFile"  
 class="ch.qos.logback.core.rolling.RollingFileAppender">  
 <file>${LOGS}/spring-boot-logger.log</file>  
 <encoder  
 class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">  
 <Pattern>%d %p %C{1.} [%t] %m%n</Pattern>  
 </encoder>  
  
 <rollingPolicy  
 class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 <!-- rollover daily and when the file reaches 10 MegaBytes -->  
 <fileNamePattern>${LOGS}/archived/spring-boot-logger-%d{yyyy-MM-dd}.%i.log  
 </fileNamePattern>  
 <timeBasedFileNamingAndTriggeringPolicy  
 class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">  
 <maxFileSize>10MB</maxFileSize>  
 </timeBasedFileNamingAndTriggeringPolicy>  
 </rollingPolicy>  
 </appender>  
  
 <!-- LOG everything at INFO level -->  
 <root level="info">  
 <appender-ref ref="RollingFile" />  
 <appender-ref ref="Console" />  
 </root>  
  
 <!-- LOG "com.baeldung\*" at TRACE level -->  
 <logger name="com.baeldung" level="trace" additivity="false">  
 <appender-ref ref="RollingFile" />  
 <appender-ref ref="Console" />  
 </logger>  
  
</configuration>

1. **Результат работы программы**



# **Практическая работа №20**

1. **Постановка задачи и персональный вариант**

Тема: Использование Spring AOP. Pointcut, JoinPoint. Advice.

Постановка задачи: для приложения из предыдущего задания добавить логирование времени выполнения каждого метода сервиса с использованием Spring AOP.

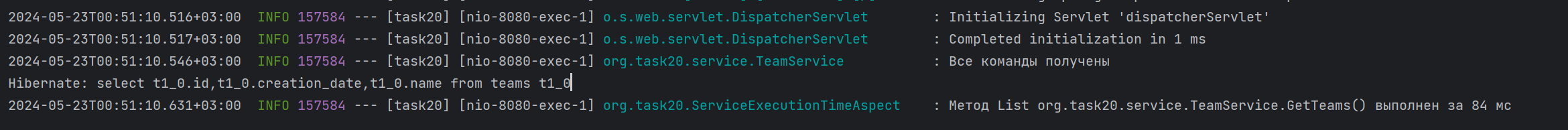
1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task20>

Файл: ServiceExecutionTimeAspect.java

package org.task20;  
  
import lombok.extern.slf4j.Slf4j;  
import org.aspectj.lang.JoinPoint;  
import org.aspectj.lang.annotation.After;  
import org.aspectj.lang.annotation.Aspect;  
import org.aspectj.lang.annotation.Before;  
import org.springframework.stereotype.Component;  
  
@Aspect  
@Component  
@Slf4j  
public class ServiceExecutionTimeAspect {  
 private long startTime;  
  
 @Before("execution(\* org.task20.service.TeamService.\*(..))")  
 public void beforeMethodExecution() {  
 startTime = System.*currentTimeMillis*();  
 }  
  
 @After("execution(\* org.task20.service.TeamService.\*(..))")  
 public void afterMethodExecution(JoinPoint joinPoint) {  
 long executionTime = System.*currentTimeMillis*() - startTime;  
 *log*.info("Метод " + joinPoint.getSignature() + " выполнен за " + executionTime + " мс");  
 }  
}

1. **Результат работы программы**



# **Практическая работа №21**

1. **Постановка задачи и персональный вариант**

Тема: Проксирование. Аннотация Transactional. Аннотация Async.

Постановка задачи: для приложения из предыдущего задания пометить все классы сервисов, в которых происходит взаимодействие с базой данных, как Transactional. Добавить отправку информации о сохранении каждого объекта по электронной почте, создав отдельный класс EmailService с асинхронными методами отправки сообщений. Для асинхронности методов используйте аннотацию Async.

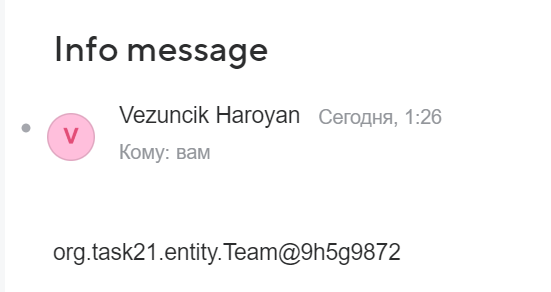
1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/task21>

Файл: EmailSender.java

package org.task21;  
  
import lombok.AllArgsConstructor;  
import org.springframework.mail.SimpleMailMessage;  
import org.springframework.mail.javamail.JavaMailSender;  
import org.springframework.scheduling.annotation.Async;  
import org.springframework.stereotype.Service;  
import org.task21.entity.Team;  
  
@Service  
@AllArgsConstructor  
public class EmailSender  
{  
 private final JavaMailSender javaMailSender;  
  
 @Async  
 public void sendEmail(Team team) {  
 SimpleMailMessage message = new SimpleMailMessage();  
  
 message.setFrom("vezuncik03@mail.ru");  
 message.setTo("vezuncik03@mail.ru");  
 message.setSubject("Info message");  
 message.setText(team.toString());  
  
 javaMailSender.send(message);  
 }  
}

1. **Результат работы программы**



# **Практическая работа №22**

1. **Постановка задачи и персональный вариант**

Тема: Планирование заданий. Scheduler в Spring.

Постановка задачи: для приложения из предыдущего задания создать класс-сервис с методом, который будет вызываться каждые 30 минут и очищать определённую директорию, а затем создавать по файлу для каждой из сущностей и загружать туда все данные из базы данных. Также добавить возможность вызывать данный метод с использованием Java Management Extensions (JMX).

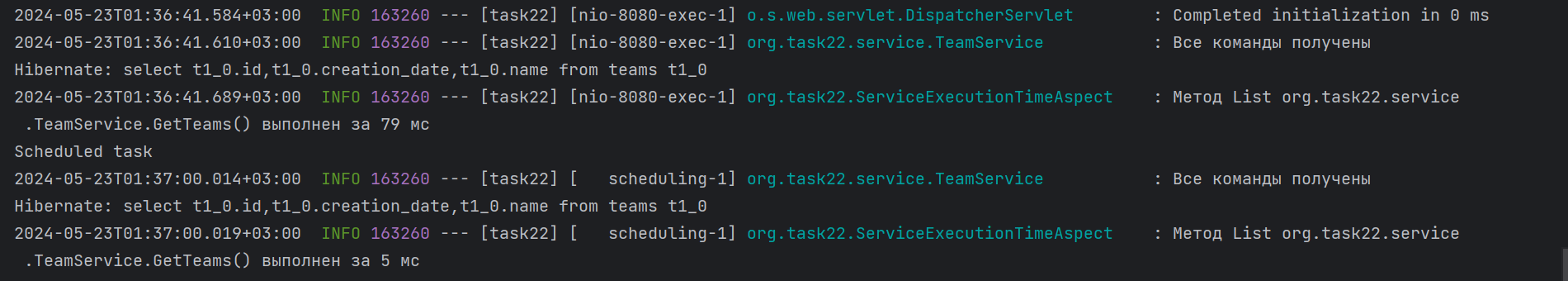
1. **Код**

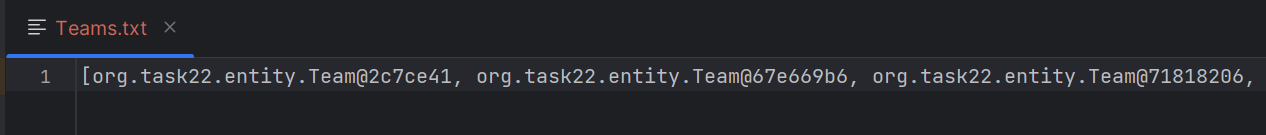
<https://github.com/Shumila71/JavaPractice/tree/main/task22>

Файл: ScheduledService.java

package org.task22.service;  
  
import lombok.AllArgsConstructor;  
  
import org.springframework.scheduling.annotation.EnableScheduling;  
  
import org.springframework.scheduling.annotation.Scheduled;  
  
import org.springframework.stereotype.Service;  
  
import java.io.File;  
  
import java.io.FileWriter;  
  
import java.io.IOException;  
  
import java.util.Objects;  
  
@Service  
@EnableScheduling  
@AllArgsConstructor  
public class ScheduledService  
{  
 private final TeamService service;  
  
 @Scheduled(cron = "0 \* \* \* \* \*")  
 public void task() throws IOException  
 {  
 String path = "D:\\code\\\_4\\java\\task22\\src\\main\\resources\\entities";  
 System.*out*.println("Scheduled task");  
  
 for (File myFile : Objects.*requireNonNull*(new File(path).listFiles()))  
 {  
 if (myFile.isFile()) myFile.delete();  
 }  
  
 File file = new File(path +"/Teams.txt");  
 file.createNewFile();  
 FileWriter fileWriter = new FileWriter(file);  
 fileWriter.write(service.GetTeams().toString());  
 fileWriter.close();  
 }  
}

1. **Результат работы программы**





# **Практическая работа №23**

1. **Постановка задачи и персональный вариант**

Тема: Использование Spring Security для аутентификации и авторизации пользователей.

Постановка задачи: В приложении из предыдущего задания добавить возможность регистрации и авторизации пользователей, хранение cookie сессий в базе данных PostgreSQL, хеширование паролей алгоритмом Bcrypt, защиту всех запросов, кроме запросов на авторизацию и регистрацию, от неавторизированных пользователей.

1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/Task23>

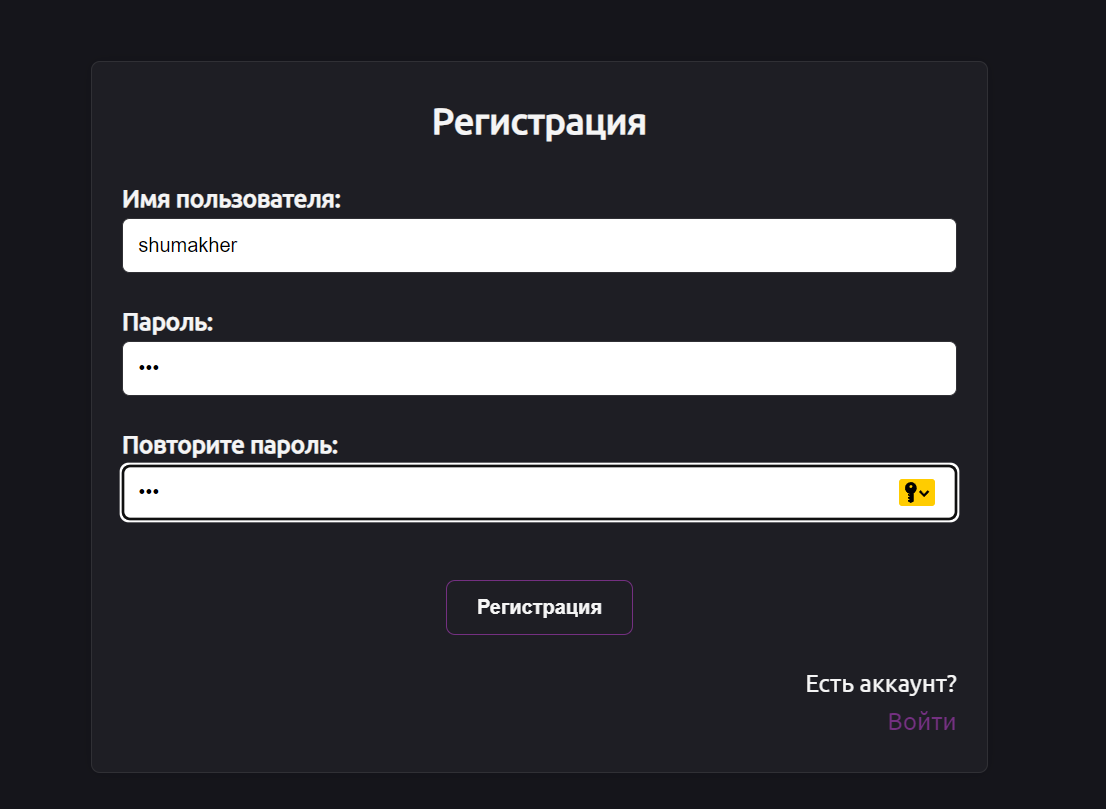
Файл: SecurityConfig.java

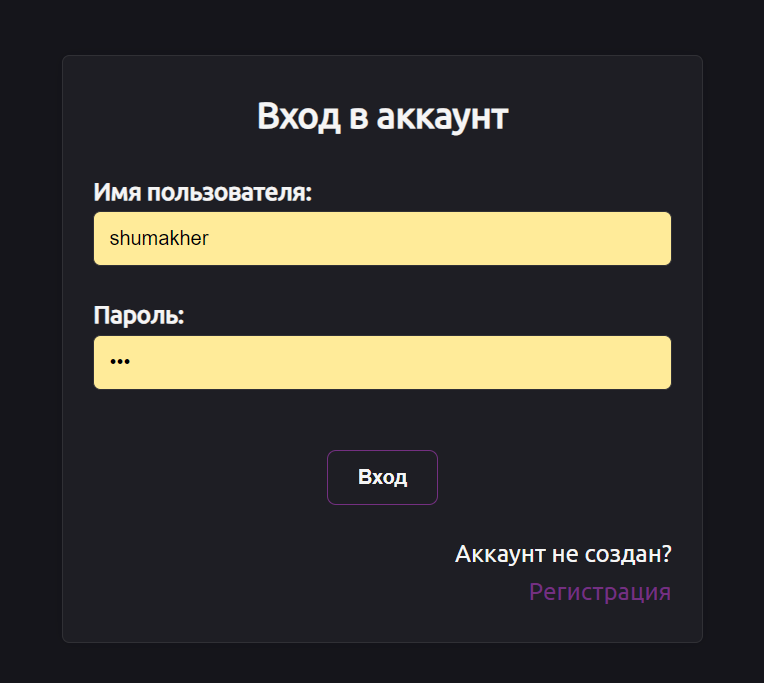
package org.task23.config;  
  
import org.task23.service.AppUserDetailsService;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.authentication.AuthenticationProvider;  
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.web.SecurityFilterChain;  
  
@Configuration  
@EnableWebSecurity  
public class SecurityConfig  
{  
 @Bean  
 public UserDetailsService userDetailsService()  
 {  
 return new AppUserDetailsService();  
 }  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception  
 {  
 return http.csrf(AbstractHttpConfigurer::disable).cors(AbstractHttpConfigurer::disable)  
 .authorizeHttpRequests((auth) ->  
 auth.requestMatchers("/login", "/register", "/", "/resources/\*\*").permitAll()  
 .anyRequest().authenticated())  
 .formLogin(form -> form  
 .loginPage("/login")  
 .defaultSuccessUrl("/catalog")  
 .permitAll())  
 .logout(logout -> logout  
 .logoutUrl("/logout")  
 .logoutSuccessUrl("/")  
 .invalidateHttpSession(true)  
 .deleteCookies("JSESSIONID", "cartItems"))  
 .build();  
 }  
  
 @Bean  
 public AuthenticationProvider authenticationProvider()  
 {  
 var provider = new DaoAuthenticationProvider();  
 provider.setUserDetailsService(userDetailsService());  
 provider.setPasswordEncoder(passwordEncoder());  
 return provider;  
 }  
  
 @Bean  
 public PasswordEncoder passwordEncoder()  
 {  
 return new BCryptPasswordEncoder();  
 }  
}

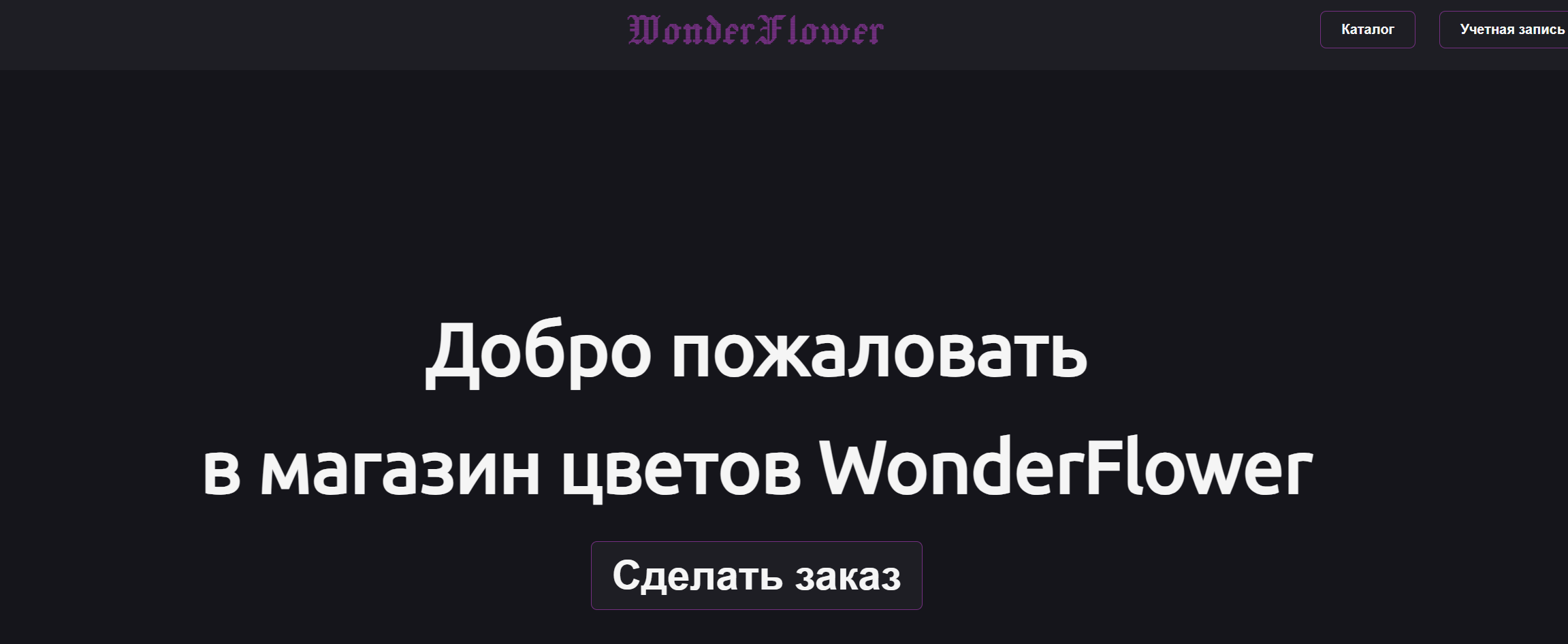
Файл: EntryController.java

package org.task23.controller;  
  
import org.task23.entity.UserDTO;  
import org.task23.service.EntryService;  
import jakarta.servlet.ServletException;  
import jakarta.servlet.http.HttpServletRequest;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.ModelAttribute;  
import org.springframework.web.bind.annotation.PostMapping;  
  
  
@Controller  
public class EntryController  
{  
 @Autowired  
 EntryService entryService;  
  
 @GetMapping("/register")  
 public String regGet(Model model)  
 {  
 model.addAttribute("user", new UserDTO());  
 return "register";  
 }  
  
 @PostMapping("/register")  
 public String regPost(@ModelAttribute("user") UserDTO newUser, HttpServletRequest request) throws ServletException  
 {  
 if (!newUser.getPassword().equals(newUser.getConfirmPassword()))  
 {  
 return "redirect:/register?error";  
 }  
  
 if (entryService.addUser(newUser))  
 {  
 request.login(newUser.getName(), newUser.getPassword());  
 return "redirect:/";  
 }  
  
 return "redirect:/register?exists";  
 }  
  
 @GetMapping("/login")  
 public String loginPage()  
 {  
 return "login";  
 }  
}

1. **Результат работы программы**







# **Практическая работа №24**

1. **Постановка задачи и персональный вариант**

Тема: Тестирование в Spring Framework с использованием Junit.

Постановка задачи: написать модульное тестирование для всех классов сервисов приложения из предыдущего задания.

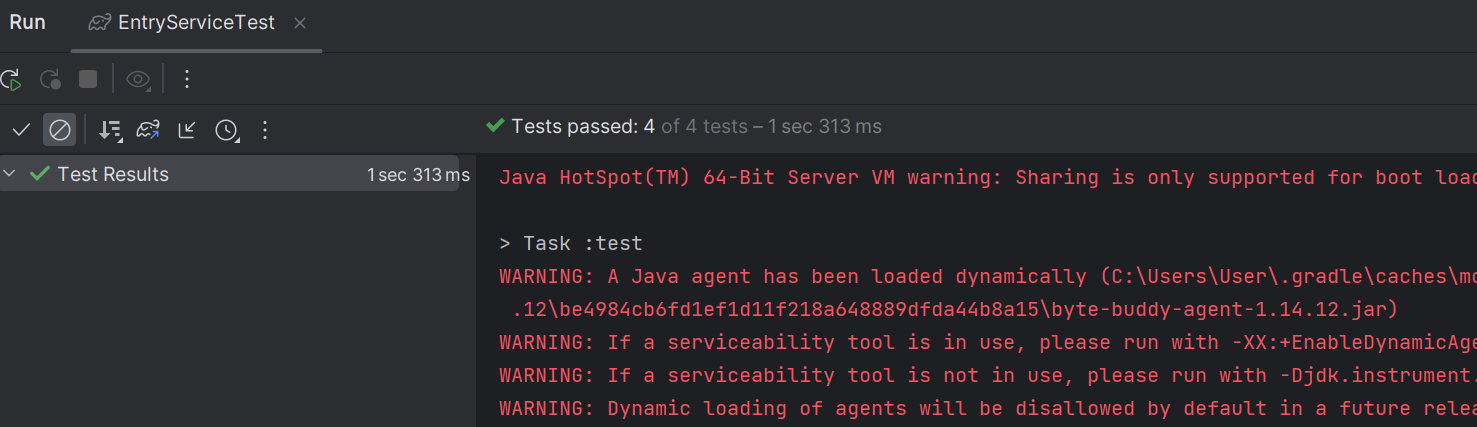
1. **Код**

<https://github.com/Shumila71/JavaPractice/tree/main/Task24>

Файл: EntryServiceTest.java

package org.task24.service;  
  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.junit.jupiter.MockitoExtension;  
import org.task24.entity.User;  
import org.task24.entity.UserDTO;  
import org.task24.repository.UserRepository;  
import org.springframework.security.crypto.password.PasswordEncoder;  
  
import java.util.Optional;  
  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.Mockito.\*;  
  
@ExtendWith(MockitoExtension.class)  
public class EntryServiceTest  
{  
 @Mock  
 private UserRepository userRepository;  
 @Mock  
 private PasswordEncoder encoder;  
 @InjectMocks  
 private EntryService entryService;  
  
 @Test  
 void testAddUser\_Successful()  
 {  
 UserDTO newUser = new UserDTO();  
 newUser.setName("testUser");  
 newUser.setPassword("password");  
 newUser.setConfirmPassword("password");  
 *when*(userRepository.existsByName(newUser.getName())).thenReturn(false);  
 *when*(encoder.encode(newUser.getPassword())).thenReturn("encodedPassword");  
  
 boolean result = entryService.addUser(newUser);  
  
 *assertTrue*(result);  
 *verify*(userRepository, *times*(1)).existsByName(newUser.getName());  
 *verify*(encoder, *times*(1)).encode(newUser.getPassword());  
 *verify*(userRepository, *times*(1)).save(*any*(User.class));  
 }  
  
 @Test  
 void testAddUser\_UserAlreadyExists()  
 {  
 UserDTO existingUser = new UserDTO();  
 existingUser.setName("testUser");  
 existingUser.setPassword("password");  
 existingUser.setConfirmPassword("password");  
 *when*(userRepository.existsByName(existingUser.getName())).thenReturn(true);  
  
 boolean result = entryService.addUser(existingUser);  
  
 *assertFalse*(result);  
 *verify*(userRepository, *times*(1)).existsByName(existingUser.getName());  
 *verify*(encoder, *never*()).encode(*anyString*());  
 *verify*(userRepository, *never*()).save(*any*(User.class));  
 }  
  
 @Test  
 void testFindUserByName\_UserExists()  
 {  
 String username = "existingUser";  
 User user = new User();  
 *when*(userRepository.findUserByName(username)).thenReturn(Optional.*of*(user));  
  
 Optional<User> result = entryService.findUserByName(username);  
  
 *assertTrue*(result.isPresent());  
 *assertEquals*(user, result.get());  
 *verify*(userRepository, *times*(1)).findUserByName(username);  
 }  
  
 @Test  
 void testFindUserByName\_UserDoesNotExist()  
 {  
 String username = "nonExistingUser";  
 *when*(userRepository.findUserByName(username)).thenReturn(Optional.*empty*());  
  
 Optional<User> result = entryService.findUserByName(username);  
  
 *assertFalse*(result.isPresent());  
 *verify*(userRepository, *times*(1)).findUserByName(username);  
 }  
}

1. **Результат работы программы**



# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Стелтинг С., Маасен О. Применение шаблонов Java. Библиотека профессионала.: Пер. с англ. — М.: Издательский дом "Вильяме", 2002. — 576 с.: ил. — Парал. тит. англ.
2. Functional Interfaces in Java: Fundamentals and Examples 1st ed. Edition, Kindle Edition [Электронный ресурс]. – URL: https://www.amazon.com/Functional-Interfaces-Java-Fundamentals-Examplesebook/dp/B07NRHQSCW (дата обращения: 29.01.21).
3. Hibernate Search 6.0.0.Final: Reference Documentation [Электронный ресурс]. – URL:https://docs.jboss.org/hibernate/stable/search/reference/enUS/html\_single/ (дата обращения: 29.01.21).
4. Паттерны проектирования на Java. Каталог Java-примеров. [Электронный ресурс]. – URL: https://refactoring.guru/ru/design-patterns/java (дата обращения: 29.01.21).
5. Руководство по Spring [Электронный ресурс]. – URL: https://proselyte.net/tutorials/spring-tutorial-full-version/ (дата обращения: 29.01.21).
6. The Reactive Manifesto [Электронный ресурс]. – URL: https://www.reactivemanifesto.org/ (дата обращения: 29.01.21).
7. Spring Framework Documentation [Электронный ресурс]. – URL: https://docs.spring.io/spring-framework/docs/current/reference/html/web.html (дата обращения: 29.01.21).
8. Hibernate Search 6.0.0. Final: Reference Documentation [Электронный ресурс]. – URL:https://docs.jboss.org/hibernate/stable/search/reference/enUS/html\_single/ (дата обращения: 29.05.24)