**Практическая работа №17**

1. **Постановка задачи и персональный вариант**

Цель работы: 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 и пояснение к коду**

Листинг 63 – код для пр. работы №17

package com.example.demo.task17.controllers;  
  
  
import com.example.demo.task17.Item;  
import com.example.demo.task17.Order;  
import com.example.demo.task17.repository.ItemRepository;  
import com.example.demo.task17.repository.OrderRepository;  
import jakarta.persistence.EntityManager;  
import jakarta.persistence.TypedQuery;  
import jakarta.persistence.criteria.CriteriaBuilder;  
import jakarta.persistence.criteria.CriteriaQuery;  
import jakarta.persistence.criteria.Predicate;  
import jakarta.persistence.criteria.Root;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
@RestController  
public class OrderController {  
 private final OrderRepository orderRepository;  
 private final ItemRepository itemRepository;  
 private final EntityManager entityManager;  
  
  
 @Autowired  
 public OrderController(OrderRepository orderRepository, ItemRepository itemRepository, EntityManager entityManager) {  
 this.orderRepository = orderRepository;  
 this.itemRepository = itemRepository;  
 this.entityManager = entityManager;  
 }  
  
 public CriteriaBuilder getCriteriaBuilder() {  
 return entityManager.getCriteriaBuilder();  
 }  
  
 public TypedQuery<Order> createQuery(CriteriaQuery<Order> criteriaQuery) {  
 return entityManager.createQuery(criteriaQuery);  
 }  
  
 @GetMapping("/orders")  
 public List<Order> getOrders(@RequestParam(required = false) LocalDate from,  
 @RequestParam(required = false) LocalDate to) {  
  
 CriteriaBuilder cb = getCriteriaBuilder();  
 CriteriaQuery<Order> query = cb.createQuery(Order.class);  
 Root<Order> root = query.from(Order.class);  
  
 List<Predicate> predicates = new ArrayList<>();  
  
 if (from != null) {  
 predicates.add(cb.greaterThanOrEqualTo(root.get("orderDate"), from));  
 }  
 if (to != null) {  
 predicates.add(cb.lessThanOrEqualTo(root.get("orderDate"), to));  
 }  
  
 if (!predicates.isEmpty()) {  
 query.where(predicates.toArray(new Predicate[0]));  
 }  
  
 return createQuery(query).getResultList();  
 }  
  
 @DeleteMapping("/orders")  
 public List<Order> deleteOrder(@RequestBody Order order){  
 orderRepository.delete(order);  
 return orderRepository.findAll();  
 }  
  
 @GetMapping("/order/{orderId}")  
 public @ResponseBody Optional<Order> getOrder(@PathVariable("orderId") Integer orderId){  
 return orderRepository.findById(orderId);  
 }  
 @PostMapping("/orders")  
 public void addOrder(@RequestBody Order order){  
 System.*out*.println(order.items);  
 order = orderRepository.save(order);  
 for ( Item item : order.items){  
 item.setOrder(order);  
 }  
 order.setItems(itemRepository.saveAll(order.items));  
 }  
}

1. **Результат работы программы**

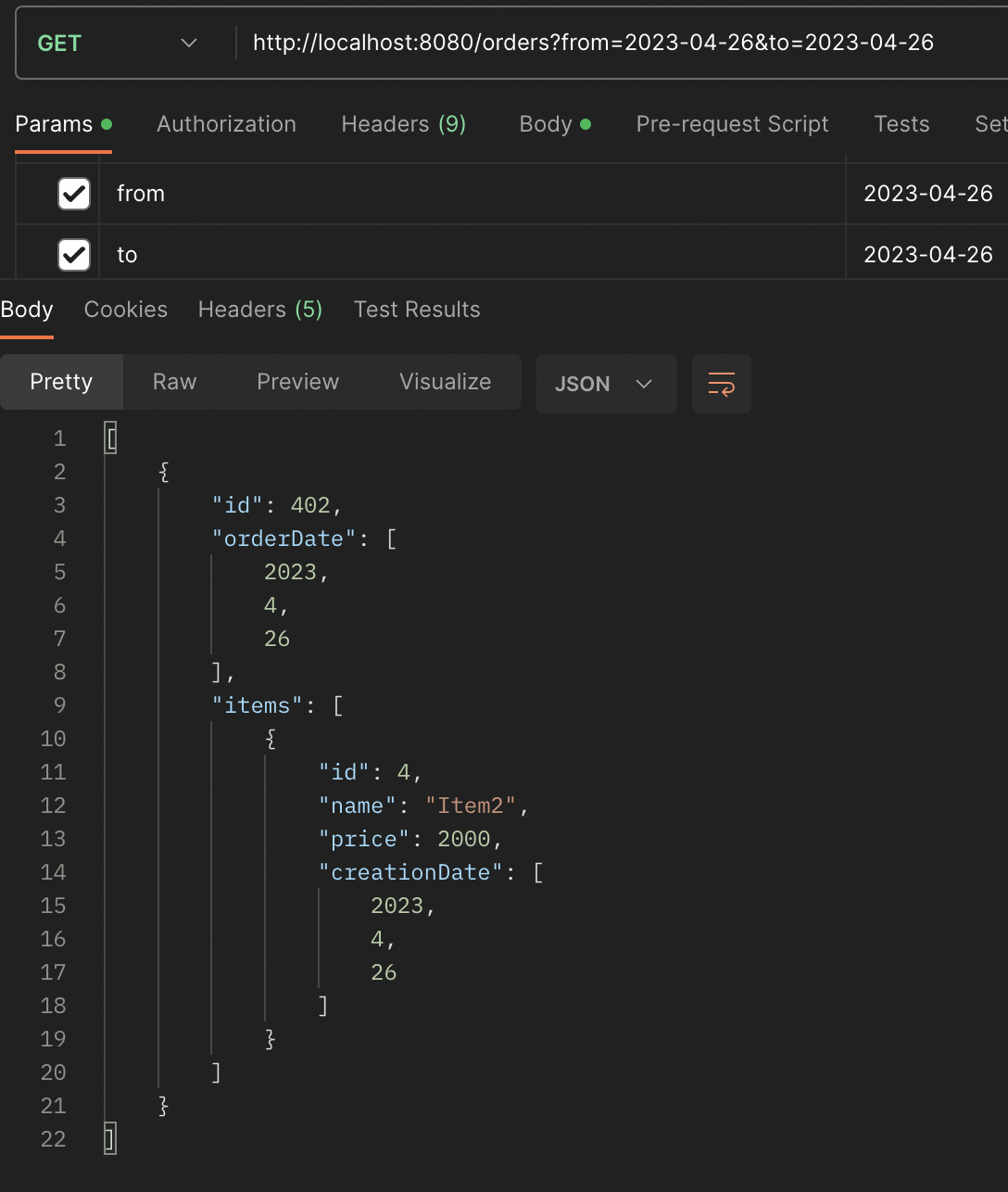
****

Рисунок 27 – Результат работы программы

**Практическая работа №18**

1. **Постановка задачи и персональный вариант**

Цель работы: 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 и пояснение к коду**

Листинг 64 – код для пр. работы №18

package com.example.demo.task18.services;  
  
import com.example.demo.task18.Item;  
  
import java.time.LocalDate;  
import java.util.List;  
import java.util.Optional;  
  
public interface ItemService {  
 Item saveItem(Item item);  
 List<Item> saveAll(List<Item> items);  
 void deleteItem(Item item);  
 Optional<Item> findById(Integer id);  
 List<Item> findAll();  
 List<Item> findAll(String name, Integer price, LocalDate creationDate);  
}

Листинг 65 – код для пр. работы №18

package com.example.demo.task18.services.impl;  
  
import com.example.demo.task18.repository.ItemRepository;  
import com.example.demo.task18.Item;  
import com.example.demo.task18.services.ItemService;  
import jakarta.persistence.TypedQuery;  
import jakarta.persistence.criteria.\*;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
public class ItemServiceImpl implements ItemService{  
  
 private final ItemRepository itemRepository;  
 private final CriteriaAPI criteriaAPI;  
  
 @Autowired  
 public ItemServiceImpl(ItemRepository itemRepository, CriteriaAPI criteriaAPI) {  
 this.itemRepository = itemRepository;  
 this.criteriaAPI = criteriaAPI;  
 }  
  
 @Override  
 public Item saveItem(Item item) {  
 return itemRepository.save(item);  
 }  
  
 @Override  
 public void deleteItem(Item item) {  
 itemRepository.delete(item);  
 }  
  
 @Override  
 public List<Item> saveAll(List<Item> items) {  
 return itemRepository.saveAll(items);  
 }  
  
 @Override  
 public Optional<Item> findById(Integer id) {  
 return itemRepository.findById(id);  
 }  
  
 @Override  
 public List<Item> findAll() {  
 return itemRepository.findAll();  
 }  
  
 @Override  
 public List<Item> findAll(String name, Integer price, LocalDate creationDate) {  
 CriteriaBuilder criteriaBuilder = criteriaAPI.getCriteriaBuilder();  
 CriteriaQuery<Item> query = criteriaBuilder.createQuery(Item.class);  
 Root<Item> root = query.from(Item.class);  
  
 List<Predicate> predicates = new ArrayList<>();  
  
 if (name != null){  
 predicates.add(criteriaBuilder.like(root.get("name"), name + "%"));  
 }  
 if (price != null){  
 predicates.add(criteriaBuilder.equal(root.get("price"), price));  
 }  
 if (creationDate != null){  
 predicates.add(criteriaBuilder.equal(root.get("creationDate"), creationDate));  
 }  
  
 query.select(root).where(predicates.toArray(new Predicate[]{}));  
 TypedQuery<Item> typedQuery = criteriaAPI.getEntityManager().createQuery(query);  
 return typedQuery.getResultList();  
 }  
  
}

Листинг 66 – код для пр. работы №18

package com.example.demo.task18.services;  
  
import com.example.demo.task18.Order;  
  
import java.time.LocalDate;  
import java.util.List;  
import java.util.Optional;  
  
public interface OrderService {  
 Order saveOrder(Order order);  
 List<Order> saveAll(List<Order> orders);  
 void deleteOrder(Order order);  
 Optional<Order> findById(Integer id);  
 List<Order> findAll();  
 List<Order> findAll(LocalDate to, LocalDate from);  
}

Листинг 67 – код для пр. работы №18

package com.example.demo.task18.services.impl;  
  
import com.example.demo.task18.Item;  
import com.example.demo.task18.repository.OrderRepository;  
import com.example.demo.task18.Order;  
import com.example.demo.task18.services.ItemService;  
import com.example.demo.task18.services.OrderService;  
import jakarta.persistence.TypedQuery;  
import jakarta.persistence.criteria.CriteriaBuilder;  
import jakarta.persistence.criteria.CriteriaQuery;  
import jakarta.persistence.criteria.Predicate;  
import jakarta.persistence.criteria.Root;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
public class OrderServiceImpl implements OrderService {  
 private final OrderRepository orderRepository;  
 private final CriteriaAPI criteriaAPI;  
 private final ItemService itemService;  
  
 @Autowired  
 public OrderServiceImpl(OrderRepository orderRepository, CriteriaAPI criteriaAPI, ItemService itemService) {  
 this.orderRepository = orderRepository;  
 this.criteriaAPI = criteriaAPI;  
 this.itemService = itemService;  
 }  
  
 @Override  
 public Order saveOrder(Order order) {  
 order = orderRepository.save(order);  
 for ( Item item : order.items){  
 item.setOrder(order);  
 }  
 order.setItems(itemService.saveAll(order.items));  
 return orderRepository.save(order);  
 }  
  
 @Override  
 public List<Order> saveAll(List<Order> orders) {  
 return orderRepository.saveAll(orders);  
 }  
  
 @Override  
 public void deleteOrder(Order order) {  
 orderRepository.delete(order);  
 }  
  
 @Override  
 public Optional<Order> findById(Integer id) {  
 return orderRepository.findById(id);  
 }  
  
 @Override  
 public List<Order> findAll() {  
 return orderRepository.findAll();  
 }  
  
 @Override  
 public List<Order> findAll(LocalDate from, LocalDate to) {  
 CriteriaBuilder criteriaBuilder = criteriaAPI.getCriteriaBuilder();  
 CriteriaQuery<Order> query = criteriaBuilder.createQuery(Order.class);  
 Root<Order> root = query.from(Order.class);  
  
 List<Predicate> predicates = new ArrayList<>();  
  
 if (from != null) {  
 predicates.add(criteriaBuilder.greaterThanOrEqualTo(root.get("orderDate"), from));  
 }  
 if (to != null) {  
 predicates.add(criteriaBuilder.lessThanOrEqualTo(root.get("orderDate"), to));  
 }  
  
 if (!predicates.isEmpty()) {  
 query.where(predicates.toArray(new Predicate[0]));  
 }  
  
 TypedQuery<Order> typedQuery = criteriaAPI.getEntityManager().createQuery(query);  
 return typedQuery.getResultList();  
 }  
}

1. **Результат работы программы**

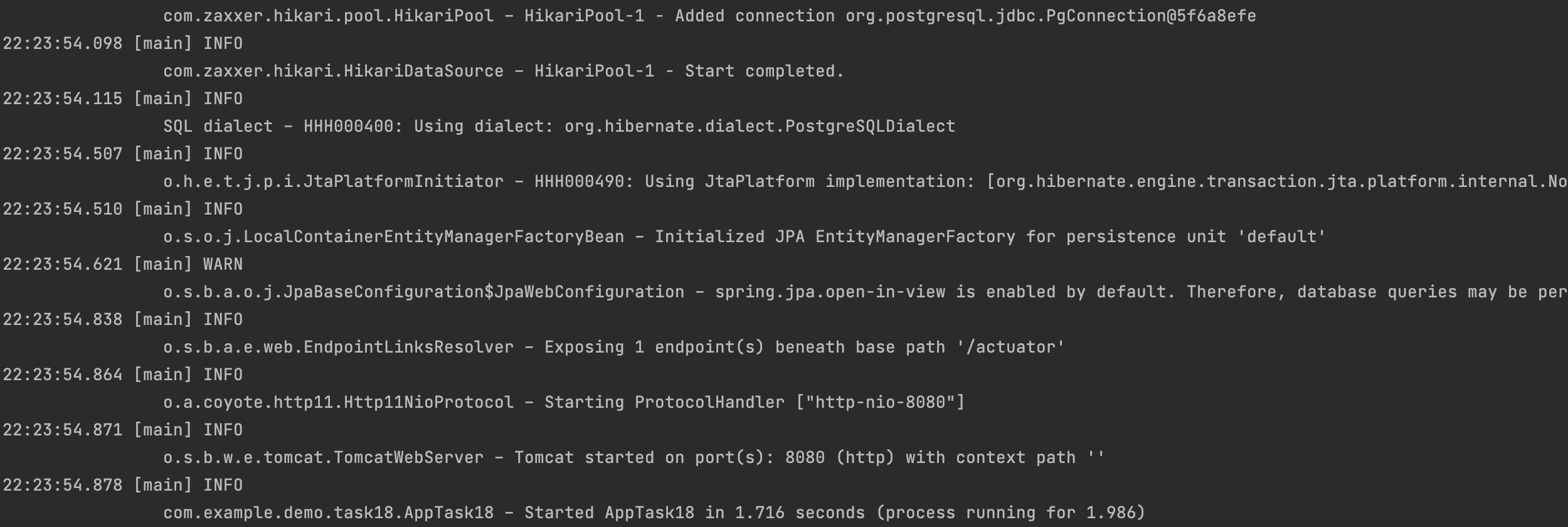
****

Рисунок 28 – Результат работы программы.

**Практическая работа №19**

1. **Постановка задачи и персональный вариант**

Цель работы: Знакомство с логированием с использованием Logback в Spring.

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

1. **Код и пояснение к коду**

Листинг 68 – код для пр. работы №19

<configuration>  
 <appender name="STDOUT"  
 class="ch.qos.logback.core.ConsoleAppender">  
 <encoder>  
 <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level  
 %logger{36} – %msg%n</pattern>  
 </encoder>  
 </appender>  
 <appender name="FILE"  
 class="ch.qos.logback.core.rolling.RollingFileAppender">  
 <file>application.log</file>  
 <rollingPolicy  
 class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">  
 <fileNamePattern>application.%d{yyyy-MM-  
 dd}.gz</fileNamePattern>  
 <maxHistory>30</maxHistory>  
 <totalSizeCap>3GB</totalSizeCap>  
 </rollingPolicy>  
 <encoder>  
 <pattern>%-4relative [%thread] %-5level %logger{35} – %msg%n</pattern>  
 </encoder>  
 </appender>  
 <root level="info">  
 <appender-ref ref="STDOUT" />  
 <appender-ref ref="FILE" />  
 </root>  
</configuration>

Листинг 69 – код для пр. работы №19

package com.example.javaframeworktemplates.task19.services.impl;  
  
import com.example.javaframeworktemplates.task19.Item;  
import com.example.javaframeworktemplates.task19.repository.ItemRepository;  
import com.example.javaframeworktemplates.task19.services.ItemService;  
import jakarta.persistence.TypedQuery;  
import jakarta.persistence.criteria.CriteriaBuilder;  
import jakarta.persistence.criteria.CriteriaQuery;  
import jakarta.persistence.criteria.Predicate;  
import jakarta.persistence.criteria.Root;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@Slf4j  
public class ItemServiceImpl implements ItemService {  
  
 private final ItemRepository itemRepository;  
 private final CriteriaAPI criteriaAPI;  
  
 @Autowired  
 public ItemServiceImpl(ItemRepository itemRepository, CriteriaAPI criteriaAPI) {  
 this.itemRepository = itemRepository;  
 this.criteriaAPI = criteriaAPI;  
 }  
  
 @Override  
 public Item saveItem(Item item) {  
 *log*.info("Saved item: {}", item);  
 return itemRepository.save(item);  
 }  
  
 @Override  
 public void deleteItem(Item item) {  
 *log*.info("Deleted item with id: {}", item.getId());  
 itemRepository.delete(item);  
 }  
  
 @Override  
 public List<Item> saveAll(List<Item> items) {  
 return itemRepository.saveAll(items);  
 }  
  
 @Override  
 public Optional<Item> findById(Integer id) {  
 *log*.info("Find item with id: {}", id);  
 return itemRepository.findById(id);  
 }  
  
 @Override  
 public List<Item> findAll() {  
 *log*.info("Find all items");  
 return itemRepository.findAll();  
 }  
  
 @Override  
 public List<Item> findAll(String name, Integer price, LocalDate creationDate) {  
 *log*.info("Find all items");  
 CriteriaBuilder criteriaBuilder = criteriaAPI.getCriteriaBuilder();  
 CriteriaQuery<Item> query = criteriaBuilder.createQuery(Item.class);  
 Root<Item> root = query.from(Item.class);  
  
 List<Predicate> predicates = new ArrayList<>();  
  
 if (name != null){  
 predicates.add(criteriaBuilder.like(root.get("name"), name + "%"));  
 }  
 if (price != null){  
 predicates.add(criteriaBuilder.equal(root.get("price"), price));  
 }  
 if (creationDate != null){  
 predicates.add(criteriaBuilder.equal(root.get("creationDate"), creationDate));  
 }  
  
 query.select(root).where(predicates.toArray(new Predicate[]{}));  
 TypedQuery<Item> typedQuery = criteriaAPI.getEntityManager().createQuery(query);  
 return typedQuery.getResultList();  
 }  
  
}

Листинг 70 – код для пр. работы №19

package com.example.javaframeworktemplates.task19.services.impl;  
  
import com.example.javaframeworktemplates.task19.Item;  
import com.example.javaframeworktemplates.task19.Order;  
import com.example.javaframeworktemplates.task19.repository.OrderRepository;  
import com.example.javaframeworktemplates.task19.services.ItemService;  
import com.example.javaframeworktemplates.task19.services.OrderService;  
import jakarta.persistence.TypedQuery;  
import jakarta.persistence.criteria.CriteriaBuilder;  
import jakarta.persistence.criteria.CriteriaQuery;  
import jakarta.persistence.criteria.Predicate;  
import jakarta.persistence.criteria.Root;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
@Service  
@Slf4j  
public class OrderServiceImpl implements OrderService {  
 private final OrderRepository orderRepository;  
 private final CriteriaAPI criteriaAPI;  
 private final ItemService itemService;  
  
 @Autowired  
 public OrderServiceImpl(OrderRepository orderRepository, CriteriaAPI criteriaAPI, ItemService itemService) {  
 this.orderRepository = orderRepository;  
 this.criteriaAPI = criteriaAPI;  
 this.itemService = itemService;  
 }  
  
 @Override  
 public Order saveOrder(Order order) {  
 order = orderRepository.save(order);  
 for ( Item item : order.items){  
 item.setOrder(order);  
 }  
 order.setItems(itemService.saveAll(order.items));  
 *log*.info("Saved order with ID: {} ", order.getId());  
 return orderRepository.save(order);  
 }  
  
 @Override  
 public List<Order> saveAll(List<Order> orders) {  
 return orderRepository.saveAll(orders);  
 }  
  
 @Override  
 public void deleteOrder(Order order) {  
 *log*.info("Deleted order with ID: {}", order.getId());  
 orderRepository.delete(order);  
 }  
  
 @Override  
 public Optional<Order> findById(Integer id) {  
 *log*.info("Find order with ID: {}", id);  
 return orderRepository.findById(id);  
 }  
  
 @Override  
 public List<Order> findAll() {  
 *log*.info("Find all orders");  
 return orderRepository.findAll();  
 }  
  
 @Override  
 public List<Order> findAll(LocalDate from, LocalDate to) {  
 *log*.info("Find all orders");  
 CriteriaBuilder criteriaBuilder = criteriaAPI.getCriteriaBuilder();  
 CriteriaQuery<Order> query = criteriaBuilder.createQuery(Order.class);  
 Root<Order> root = query.from(Order.class);  
  
 List<Predicate> predicates = new ArrayList<>();  
  
 if (from != null) {  
 predicates.add(criteriaBuilder.greaterThanOrEqualTo(root.get("orderDate"), from));  
 }  
 if (to != null) {  
 predicates.add(criteriaBuilder.lessThanOrEqualTo(root.get("orderDate"), to));  
 }  
  
 if (!predicates.isEmpty()) {  
 query.where(predicates.toArray(new Predicate[0]));  
 }  
  
 TypedQuery<Order> typedQuery = criteriaAPI.getEntityManager().createQuery(query);  
 return typedQuery.getResultList();  
 }  
}

1. **Результат работы программы**

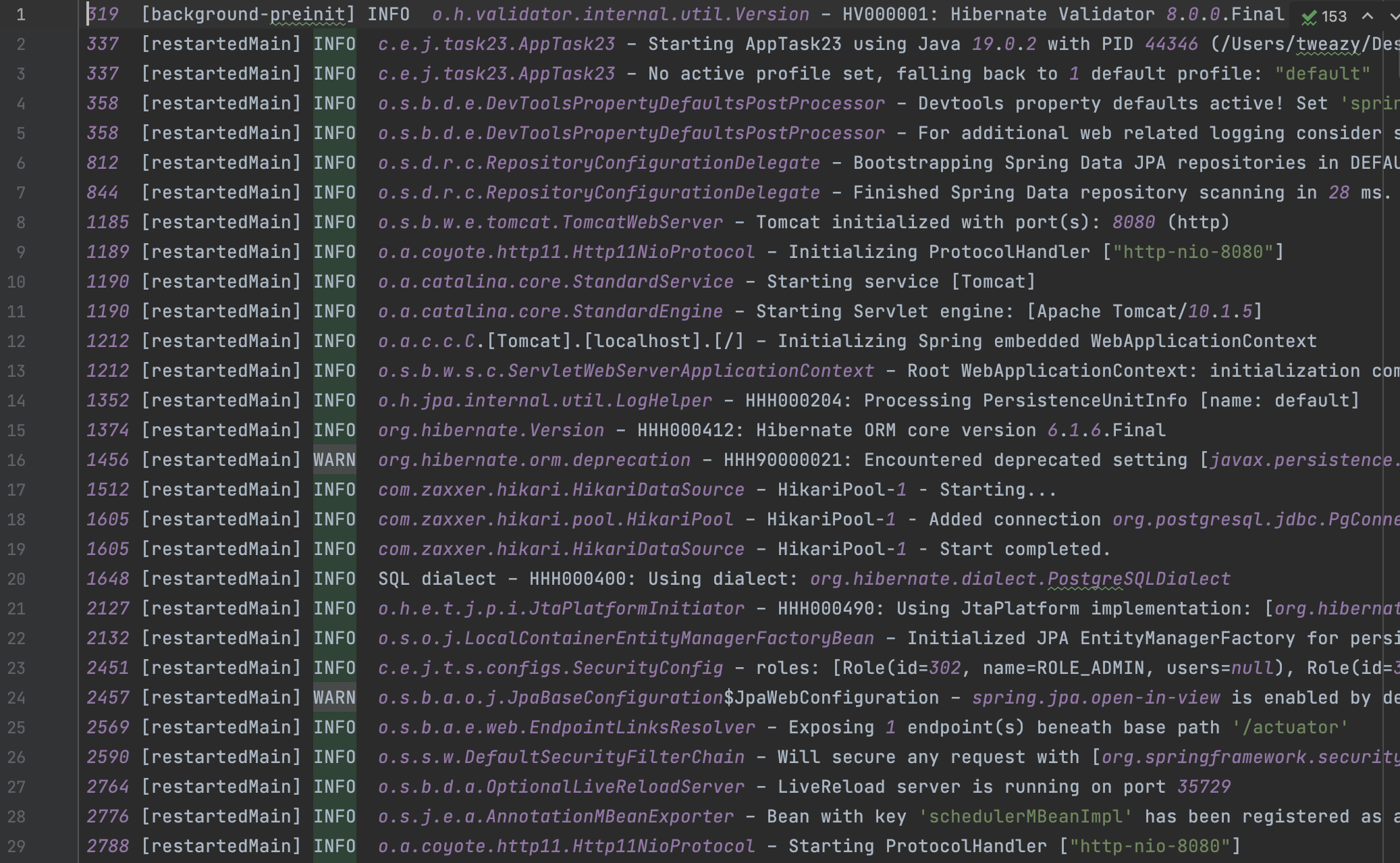
****

Рисунок 29 – Результат работы программы

**Практическая работа №20**

1. **Постановка задачи и персональный вариант**

Цель работы: 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 и пояснение к коду**

Листинг 71 – код для пр. работы №20 добавлен класс aspect

package com.example.javaframeworktemplates.task20;  
  
  
import lombok.extern.slf4j.Slf4j;  
import org.aspectj.lang.JoinPoint;  
import org.aspectj.lang.annotation.After;  
import org.aspectj.lang.annotation.Before;  
import org.aspectj.lang.annotation.Pointcut;  
import org.springframework.stereotype.Component;  
  
@Slf4j  
@Component  
@org.aspectj.lang.annotation.Aspect  
public class Aspect {  
 private long startTime;  
  
 @Before("allServiceMethods()()")  
 public void logMethod(JoinPoint joinPoint) {  
 startTime = System.*nanoTime*();  
 *log*.info("The method is now executed: " + joinPoint.getSignature());  
 }  
 @After("allServiceMethods()()")  
 public void endMethod(JoinPoint joinPoint) {  
 *log*.info("The method "+ joinPoint.getSignature() +" ended in:" +  
 ((System.*nanoTime*() - startTime) / 1\_000\_000\_000) +" ms");  
  
 }  
 @Pointcut("within(com.example.javaframeworktemplates.task20.services.impl.\*)")  
 public void allServiceMethods(){}  
}

1. **Результат работы программы**

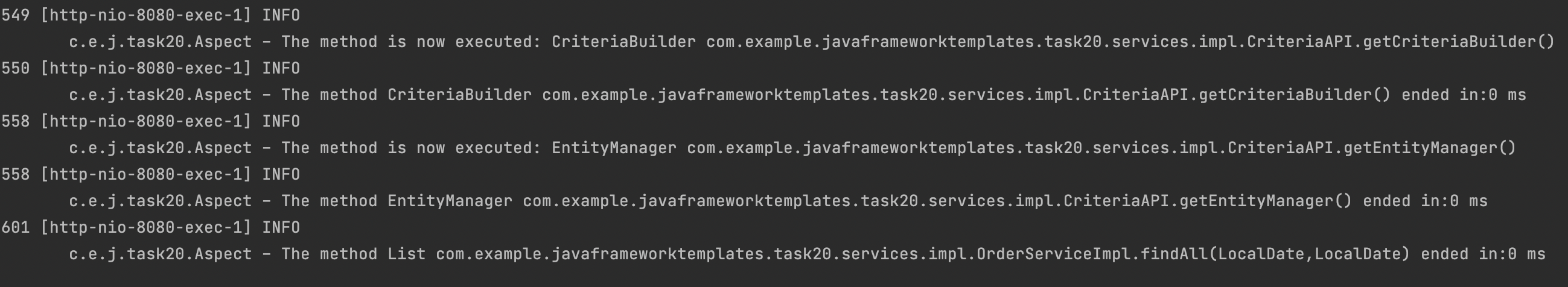
****

Рисунок 30 – Результат работы программы

**Практическая работа №21**

1. **Постановка задачи и персональный вариант**

Цель работы: 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 и пояснение к коду**

Листинг 72 – код для пр. работы №21

package com.example.javaframeworktemplates.task21.services;  
  
public interface EmailService {  
 void sendEmail(String emailTo, String subject, String message);  
}

Листинг 73 – код для пр. работы №21

package com.example.javaframeworktemplates.task21.services.impl;  
  
import com.example.javaframeworktemplates.task21.services.EmailService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.mail.SimpleMailMessage;  
import org.springframework.mail.javamail.JavaMailSender;  
import org.springframework.scheduling.annotation.Async;  
import org.springframework.stereotype.Service;  
  
@Service  
public class EmailServiceImpl implements EmailService {  
  
 @Value("${spring.mail.username}")  
 private String username;  
 private final JavaMailSender mailSender;  
  
 @Autowired  
 public EmailServiceImpl(JavaMailSender mailSender){  
 this.mailSender = mailSender;  
 }  
  
 @Async  
 @Override  
 public void sendEmail(String emailTo, String subject, String message) {  
 SimpleMailMessage mailMessage = new SimpleMailMessage();  
 mailMessage.setFrom(username);  
 mailMessage.setTo(emailTo);  
 mailMessage.setSubject(subject);  
 mailMessage.setText(message);  
  
 mailSender.send(mailMessage);  
 }  
}

Листинг 74 – код для пр. работы №21

spring.mail.host=smtp.mail.ru  
spring.mail.username=zhavoronkov03@inbox.ru  
spring.mail.password=  
spring.mail.port=465  
spring.mail.protocol=smtps

1. **Результат работы программы**

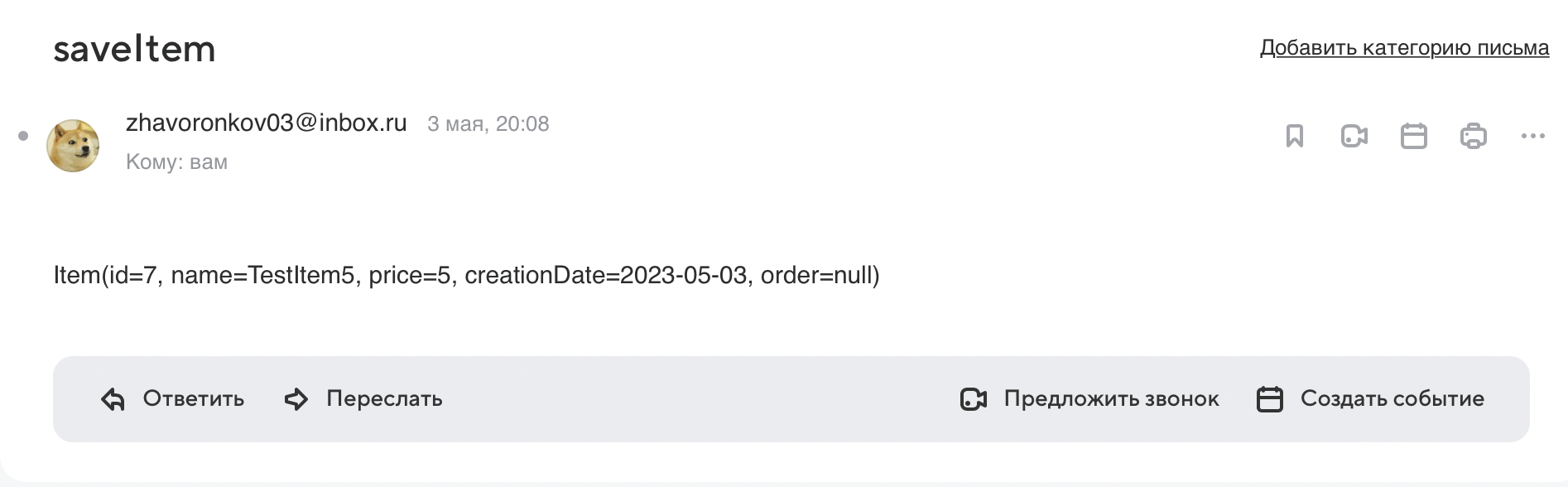
****

Рисунок 31 – Результат работы программы

**Практическая работа №22**

1. **Постановка задачи и персональный вариант**

Цель работы: 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 и пояснение к коду**

Листинг 75 – код для пр. работы №22

package com.example.javaframeworktemplates.task22.services;  
  
import java.io.IOException;  
  
public interface SchedulerService {  
 void cleanDirectory() throws IOException;  
}

Листинг 76 – код для пр. работы №22

package com.example.javaframeworktemplates.task22.services.impl;  
  
import com.example.javaframeworktemplates.task22.entities.Item;  
import com.example.javaframeworktemplates.task22.entities.Order;  
import com.example.javaframeworktemplates.task22.services.ItemService;  
import com.example.javaframeworktemplates.task22.services.OrderService;  
import com.example.javaframeworktemplates.task22.services.SchedulerService;  
import lombok.extern.slf4j.Slf4j;  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.scheduling.annotation.Scheduled;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.io.FileWriter;  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.util.List;  
  
@Service  
@Slf4j  
public class SchedulerServiceImpl implements SchedulerService {  
 private final OrderService orderService;  
 private final ItemService itemService;  
  
  
 private final Path orderSrc = Path.*of*("/Users/tweazy/Desktop/order.txt");  
 private final Path itemSrc = Path.*of*("/Users/tweazy/Desktop/item.txt");  
  
 @Autowired  
 public SchedulerServiceImpl(OrderService orderService, ItemService itemService){  
 this.itemService = itemService;  
 this.orderService = orderService;  
 }  
  
 @Scheduled(cron = "0/5 \* \* \* \* \*")  
 public void cleanDirectory() throws IOException {  
 *log*.info("cleanDirectory started");  
  
 if(Files.*exists*(orderSrc)){  
 Files.*delete*(orderSrc);  
 }  
 if(Files.*exists*(itemSrc)){  
 Files.*delete*(itemSrc);  
 }  
  
 List<Item> items = itemService.findAll();  
 List<Order> orders = orderService.findAll();  
  
 StringBuilder ordersData = new StringBuilder();  
 StringBuilder itemsData = new StringBuilder();  
  
  
 for (Order order : orders){  
 ordersData.append(" ")  
 .append(order.getId())  
 .append(" ")  
 .append(order.getOrderDate())  
 .append(" ")  
 .append(order.getItems())  
 .append("\n");  
 }  
 for (Item item : items){  
 itemsData.append(item.getId())  
 .append(" ")  
 .append(item.getName())  
 .append(" ")  
 .append(item.getPrice())  
 .append(" ")  
 .append(item.getCreationDate())  
 .append("\n");  
 }  
 FileWriter orderWriter = new FileWriter("/Users/tweazy/Desktop/order.txt");  
 FileWriter itemWriter = new FileWriter("/Users/tweazy/Desktop/item.txt");  
  
 orderWriter.write(ordersData.toString());  
 orderWriter.close();  
  
 itemWriter.write(itemsData.toString());  
 itemWriter.close();  
 }  
}

Листинг 77 – код для пр. работы №22

package com.example.javaframeworktemplates.task22.jmx;  
  
import java.io.IOException;  
  
public interface SchedulerMBean {  
 void backupData() throws IOException;  
}

Листинг 78 – код для пр. работы №22

package com.example.javaframeworktemplates.task22.jmx.impl;  
  
import com.example.javaframeworktemplates.task22.jmx.SchedulerMBean;  
import com.example.javaframeworktemplates.task22.services.SchedulerService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.jmx.export.annotation.ManagedAttribute;  
import org.springframework.jmx.export.annotation.ManagedResource;  
import org.springframework.stereotype.Component;  
  
import java.io.IOException;  
  
@Component  
@ManagedResource(description = "Create backup")  
public class SchedulerMBeanImpl implements SchedulerMBean {  
 private final SchedulerService schedulerService;  
  
 @Autowired  
 public SchedulerMBeanImpl(SchedulerService schedulerService){  
 this.schedulerService = schedulerService;  
 }  
  
 @Override  
 public void backupData() throws IOException {  
 schedulerService.cleanDirectory();  
 }  
}

1. **Результат работы программы**

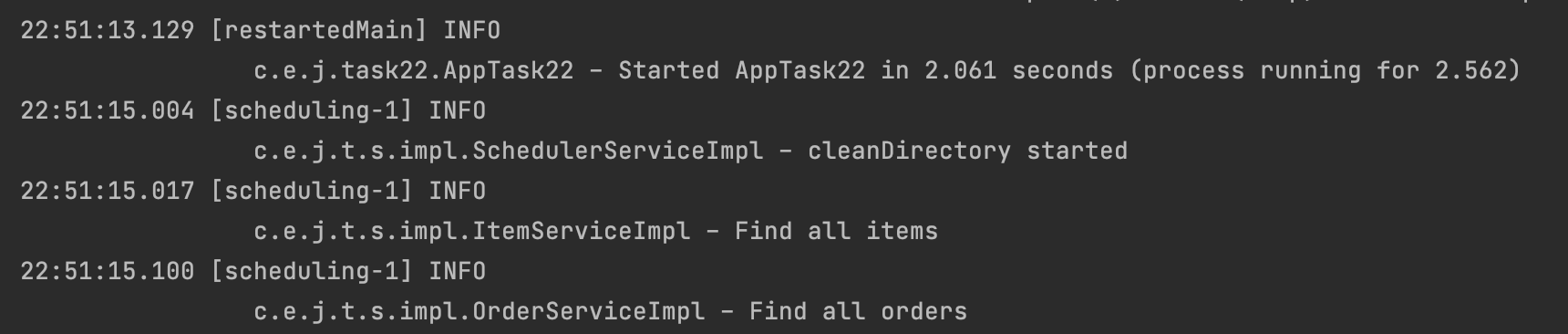
****

Рисунок 32 – Результат работы программы

**Практическая работа №23**

1. **Постановка задачи и персональный вариант**

Цель работы: 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 и пояснение к коду**

Листинг 79 – код для пр. работы №23

package com.example.javaframeworktemplates.task23.security.custom.models;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import lombok.RequiredArgsConstructor;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
  
import java.util.Collection;  
import java.util.Set;  
  
@Data  
@Entity  
@AllArgsConstructor  
@NoArgsConstructor  
@Table(name = "users")  
public class User implements UserDetails {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private String username;  
 private String firstname;  
 private String lastname;  
 private String email;  
 private String password;  
 @ManyToMany(fetch = FetchType.*EAGER*, cascade = {CascadeType.*REFRESH*})  
 private Set<Role> roles;  
 public User(String username,String email, String password, Set<Role> roles) {  
 this.username = username;  
 this.email = email;  
 this.password = password;  
 this.roles = roles;  
 }  
  
 public User(String username, String firstname, String lastname, String email, String password) {  
 this.username = username;  
 this.firstname = firstname;  
 this.lastname = lastname;  
 this.email = email;  
 this.password = password;  
 }  
  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 return getRoles();  
 }  
  
 @Override  
 public boolean isAccountNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return true;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return true;  
 }  
}

Листинг 80 – код для пр. работы №23

package com.example.javaframeworktemplates.task23.security.custom.models;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import org.springframework.security.core.GrantedAuthority;  
  
import java.util.Set;  
  
@Data  
@Entity  
@AllArgsConstructor  
@NoArgsConstructor  
@Table(name = "roles")  
public class Role implements GrantedAuthority {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 private Long id;  
 private String name;  
 @ManyToMany(mappedBy = "roles", fetch = FetchType.*EAGER*, cascade = {CascadeType.*REFRESH*})  
 @Transient  
 private Set<User> users;  
  
 public Role(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public String getAuthority() {  
 return getName();  
 }  
}

Листинг 81 – код для пр. работы №23

package com.example.javaframeworktemplates.task23.security.custom.services.impl;  
  
import com.example.javaframeworktemplates.task23.security.custom.models.Role;  
import com.example.javaframeworktemplates.task23.security.custom.repositories.RoleRepository;  
import com.example.javaframeworktemplates.task23.security.custom.services.RoleService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
@Service  
public class RoleServiceImpl implements RoleService {  
 private final RoleRepository roleRepository;  
 @Autowired  
 public RoleServiceImpl(RoleRepository roleRepository) {  
 this.roleRepository = roleRepository;  
 }  
  
 @Override  
 public Role findByName(String name) {  
 return roleRepository.findByName(name);  
 }  
  
 @Override  
 public Role saveRole(Role role) {  
 return roleRepository.save(role);  
 }  
  
 @Override  
 public List<Role> findAllRoles() {  
 return roleRepository.findAll();  
 }  
  
 @Override  
 public List<Role> saveAllRoles(Iterable<Role> roles) {  
 return roleRepository.saveAll(roles);  
 }  
}

Листинг 82 – код для пр. работы №23

package com.example.javaframeworktemplates.task23.security.custom.services.impl;  
  
  
import com.example.javaframeworktemplates.task23.security.custom.models.User;  
import com.example.javaframeworktemplates.task23.security.custom.models.enumerates.ERole;  
import com.example.javaframeworktemplates.task23.security.custom.repositories.UserRepository;  
import com.example.javaframeworktemplates.task23.security.custom.services.RoleService;  
import com.example.javaframeworktemplates.task23.security.custom.services.UserService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.stereotype.Service;  
  
import java.util.Collections;  
import java.util.List;  
  
  
@Service  
public class UserServiceImpl implements UserService {  
 private final UserRepository userRepository;  
 private final RoleService roleService;  
 private final PasswordEncoder passwordEncoder;  
  
 @Autowired  
 public UserServiceImpl(  
 UserRepository userRepository,  
 RoleService roleService,  
 PasswordEncoder passwordEncoder  
 ) {  
 this.userRepository = userRepository;  
 this.roleService = roleService;  
 this.passwordEncoder = passwordEncoder;  
 }  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 User user = userRepository.findByUsername(username);  
 if (user == null) {  
 throw new UsernameNotFoundException("User not found");  
 }  
 return user;  
 }  
  
 @Override  
 public User findByUsername(String username) {  
 return userRepository.findByUsername(username);  
 }  
  
 @Override  
 public User saveUser(User user) {  
 if (userRepository.findByUsername(user.getUsername()) != null) {  
 return null;  
 }  
 if (user.getRoles() == null || user.getRoles().size() == 0) {  
 user.setRoles(Collections.*singleton*(roleService.findByName(ERole.*ROLE\_USER*.toString())));  
 }  
 user.setPassword(passwordEncoder.encode(user.getPassword()));  
 return userRepository.save(user);  
 }  
  
 @Override  
 public List<User> findAllUsers() {  
 return userRepository.findAll();  
 }  
  
 @Override  
 public List<User> saveAllUsers(Iterable<User> users) {  
 return userRepository.saveAll(users);  
 }  
}

Листинг 83 – код для пр. работы №23

package com.example.javaframeworktemplates.task23.security.controllers;  
  
import com.example.javaframeworktemplates.task23.security.custom.models.User;  
import com.example.javaframeworktemplates.task23.security.custom.services.UserService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
  
@Controller  
public class AuthController {  
 private final UserService userService;  
  
 @Autowired  
 public AuthController(UserService userService) {  
 this.userService = userService;  
 }  
  
 @GetMapping("/register")  
 public String register() {  
 return "views/register";  
 }  
  
 @PostMapping("/register")  
 public String registerPost(  
 @RequestParam("username") String username,  
 @RequestParam("firstname") String firstname,  
 @RequestParam("lastname") String lastname,  
 @RequestParam("email") String email,  
 @RequestParam("password") String password  
 ) {  
 userService.saveUser(new User(  
 username,  
 firstname,  
 lastname,  
 email,  
 password  
 ));  
  
 return "redirect:login";  
 }  
}

Листинг 84 – код для пр. работы №23

package com.example.javaframeworktemplates.task23.security.configs;  
  
import com.example.javaframeworktemplates.task23.security.custom.models.Role;  
import com.example.javaframeworktemplates.task23.security.custom.models.User;  
import com.example.javaframeworktemplates.task23.security.custom.models.enumerates.ERole;  
import com.example.javaframeworktemplates.task23.security.custom.services.RoleService;  
import com.example.javaframeworktemplates.task23.security.custom.services.UserService;  
import jakarta.annotation.PostConstruct;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.Customizer;  
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;  
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.password.PasswordEncoder;  
import org.springframework.security.web.SecurityFilterChain;  
  
import java.util.ArrayList;  
import java.util.HashSet;  
import java.util.List;  
  
@Slf4j  
@Configuration  
@EnableWebSecurity  
public class SecurityConfig {  
 private final UserDetailsService userDetailsService;  
 private final PasswordEncoder passwordEncoder;  
 private final UserService userService;  
 private final RoleService roleService;  
  
 @Autowired  
 public SecurityConfig(  
 UserDetailsService userDetailsService,  
 PasswordEncoder passwordEncoder,  
 UserService userService,  
 RoleService roleService  
 ) {  
 this.userDetailsService = userDetailsService;  
 this.passwordEncoder = passwordEncoder;  
 this.userService = userService;  
 this.roleService = roleService;  
 }  
  
 @Autowired  
 public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {  
 auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder);  
 }  
  
 @Bean  
 public UserDetailsService userDetailsService() {  
 return userDetailsService;  
 }  
  
 @Bean  
 public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
 http  
 .cors(AbstractHttpConfigurer::disable)  
 .csrf(AbstractHttpConfigurer::disable)  
 .authorizeHttpRequests(authorize -> {  
 try {  
 authorize  
 .requestMatchers("/login").permitAll()  
 .requestMatchers("/register").permitAll()  
 .anyRequest().authenticated()  
 .and()  
 .formLogin(Customizer.*withDefaults*())  
 .logout(Customizer.*withDefaults*());  
 } catch (Exception e) {  
 throw new RuntimeException(e);  
 }  
 }  
 );  
 return http.build();  
 }  
  
 @PostConstruct  
 public void initialization() {  
 List<Role> roles = roleService.saveAllRoles(new ArrayList<>() {{  
 add(new Role(ERole.*ROLE\_ADMIN*.toString()));  
 add(new Role(ERole.*ROLE\_USER*.toString()));  
 add(new Role(ERole.*ROLE\_PUBLISHER*.toString()));  
 }});  
 User admin = userService.saveUser(new User(  
 "admin",  
 "admin@mail.ru",  
 "password",  
 new HashSet<>(roleService.findAllRoles())  
 ));  
  
 *log*.info("roles: {}, admin: {}", roles, admin);  
 }  
}

1. **Результат работы программы**

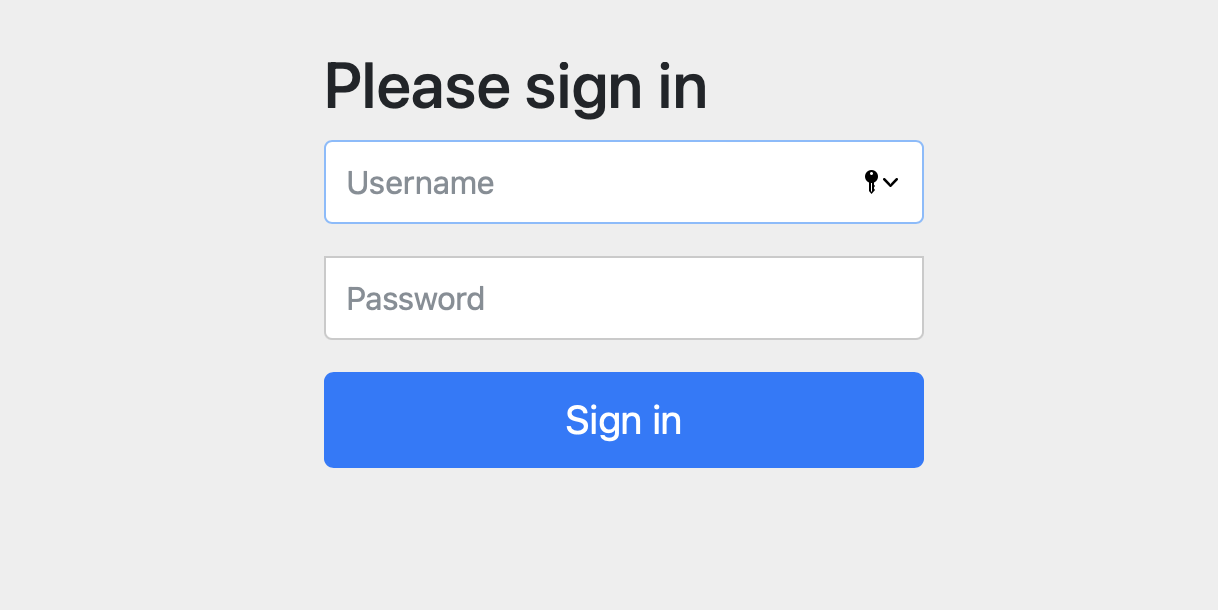
****

Рисунок 33 – Результат работы программы

**Практическая работа №24**

1. **Постановка задачи и персональный вариант**

Цель работы: Тестирование в Spring Framework с использованием Junit.

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

1. **Код и пояснение к коду**

Листинг 85 – код для пр. работы №24

package com.example.javaframeworktemplates;  
  
import com.example.javaframeworktemplates.task23.entities.Item;  
import com.example.javaframeworktemplates.task23.repository.ItemRepository;  
import com.example.javaframeworktemplates.task23.services.EmailService;  
import com.example.javaframeworktemplates.task23.services.impl.ItemServiceImpl;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.junit.jupiter.MockitoExtension;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
import static org.mockito.ArgumentMatchers.*anyString*;  
import static org.mockito.ArgumentMatchers.*eq*;  
import static org.mockito.Mockito.*verify*;  
import static org.mockito.Mockito.*when*;  
  
@ExtendWith(MockitoExtension.class)  
public class ItemServiceImplTest {  
  
 @InjectMocks  
 private ItemServiceImpl itemService;  
  
 @Mock  
 private ItemRepository itemRepository;  
  
 @Mock  
 private EmailService emailService;  
  
 @Test  
 public void findAll(){  
 Item item = new Item();  
 item.setId(35);  
 item.setName("item1");  
 item.setPrice(300);  
 item.setCreationDate(LocalDate.*of*(2023, 5, 30));  
  
 List<Item> items = new ArrayList<>();  
 items.add(item);  
  
 *when*(itemRepository.findAll()).thenReturn(items);  
  
 List<Item> retrievedItems = itemService.findAll();  
 *assertEquals*(items, retrievedItems);  
 }  
  
 @Test  
 public void deleteItem() {  
 Item item = new Item();  
 item.setId(35);  
 item.setName("item1");  
 item.setPrice(300);  
 item.setCreationDate(LocalDate.*of*(2023, 5, 30));  
  
 itemService.saveItem(item);  
  
 ResponseEntity<?> deleteResponse = itemService.deleteItem(item);  
  
 *verify*(emailService).sendEmail(*eq*("zhavoronkov03@inbox.ru") ,*eq*("saveItem"), *anyString*());  
 *assertEquals*(HttpStatus.*NOT\_FOUND*, deleteResponse.getStatusCode());  
 }  
 @Test  
 public void findById(){  
 Item item = new Item();  
 item.setId(35);  
 item.setName("item1");  
 item.setPrice(300);  
 item.setCreationDate(LocalDate.*of*(2023, 5, 30));  
  
 *when*(itemRepository.findById(35)).thenReturn(Optional.*of*(item));  
  
 ResponseEntity<Item> responseEntity = itemService.findById(35);  
  
  
 *assertEquals*(item, responseEntity.getBody());  
  
  
 }  
   
}

Листинг 86 – код для пр. работы №24

package com.example.javaframeworktemplates;  
  
import com.example.javaframeworktemplates.task23.entities.Item;  
import com.example.javaframeworktemplates.task23.entities.Order;  
import com.example.javaframeworktemplates.task23.repository.OrderRepository;  
import com.example.javaframeworktemplates.task23.services.EmailService;  
import com.example.javaframeworktemplates.task23.services.impl.OrderServiceImpl;  
import org.aspectj.weaver.ast.Or;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.junit.jupiter.MockitoExtension;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
  
import java.time.LocalDate;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.ArgumentMatchers.*anyString*;  
import static org.mockito.ArgumentMatchers.*eq*;  
import static org.mockito.Mockito.*verify*;  
import static org.mockito.Mockito.*when*;  
  
@ExtendWith(MockitoExtension.class)  
public class OrderServiceImplTest {  
  
 @InjectMocks  
 private OrderServiceImpl orderService;  
  
 @Mock  
 private OrderRepository orderRepository;  
  
 @Mock  
 private EmailService emailService;  
  
 @Test  
 public void findAll(){  
 Order order = new Order();  
 order.setId(35);  
 order.setOrderDate(LocalDate.*of*(2023, 5, 30));  
  
 List<Order> orders = new ArrayList<>();  
 orders.add(order);  
  
 *when*(orderRepository.findAll()).thenReturn(orders);  
  
 List<Order> retrievedOrders = orderService.findAll();  
 *assertEquals*(orders, retrievedOrders);  
 }  
  
 @Test  
 public void deleteOrder(){  
 Order order = new Order();  
 order.setId(35);  
 order.setOrderDate(LocalDate.*of*(2023, 5, 30));  
  
 orderService.saveOrder(order);  
  
 ResponseEntity<?> deleteResponse = orderService.deleteOrder(order);  
  
 *verify*(emailService).sendEmail(*eq*("zhavoronkov03@inbox.ru") ,*eq*("saveOrder"), *anyString*());  
 *assertEquals*(HttpStatus.*NOT\_FOUND*, deleteResponse.getStatusCode());  
 }  
  
 @Test  
 public void findById(){  
 Order order = new Order();  
 order.setId(35);  
 order.setOrderDate(LocalDate.*of*(2023, 5, 30));  
  
 *when*(orderRepository.findById(35)).thenReturn(Optional.*of*(order));  
  
 ResponseEntity<Order> responseEntity = orderService.findById(35);  
  
 *assertEquals*(order, responseEntity.getBody());  
  
 }  
}

1. **Результат работы программы**

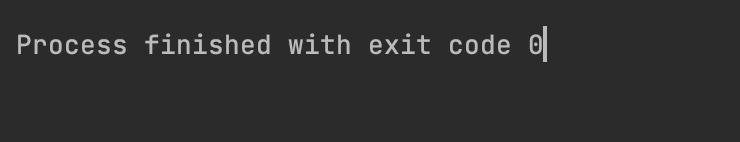
****

Рисунок 34 – Результат работы программы

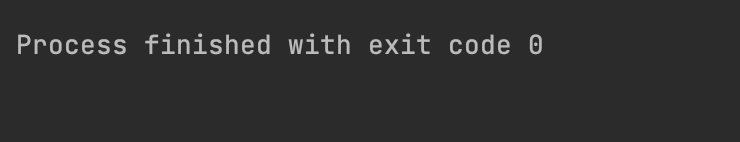
****

Рисунок 35 – Результат работы программы  
  
  
  
ОТЧЕТ  
Уважаемый, Сергей Романович, я отчёт загружу чуть-чуть позже!