Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

# институт

                   Кафедра «Программной инженерии»

# кафедра

**Практическая работа 8**

Spring JMS

# тема

Преподаватель \_\_\_\_\_\_\_\_ А. С. Кузнецов

# подпись, дата инициалы, фамилия

Студент ЗКИ21-16б \_\_\_\_\_\_\_\_ Л. Д. Чумаченко

# номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2025

### 1 Задание

Общая постановка задачи.

Изменить приложение из практического задания №7, №6 или №5 (на усмотрение студента) и добавить следующие возможности (пункты со "снежинкой" желательны, но не обязательны):

1) Настроить очередь (Для *ActiveMQ* или любого другого брокера сообщений *JMS*) приема сообщений для администратора.

2) При выполнении операций добавления, удаления или редактирования ресурса через *REST API* / форму создавать соответствующие уведомления и отправлять их в очередь.

3) Любым удобным способом (можно через терминал) продемонстрировать извлечение административных сообщений о выполненных операциях (из п.2).

4) Добавить кнопку-ссылку «купить» на форме. После этого в брокер сообщений отправляется сообщение о том, какой «товар»/сущность хочет купить пользователь.

5\*) В п.4 "товар" помечается как купленный и не будет показан в общем списке товаров. Необходимо добавить соответствующий столбец, или просто удалить запись о купленном товаре из БД, но перед этим не забыть отправить информацию о товаре в брокер сообщений.

6\*) Реализовать приложение-сервис приемки сообщений, которое принимает сообщение и на основе содержимого сообщения отправляет e-mail администратору по некоторому адресу (можно использовать константную строку вашего почтового ящика в домене СФУ) о том, что у него хотят купить товар.

### 2 Ход выполнения

Возьмём за основу Практическую работу 7(без механизма авторизации). В данной работе выполнены первые 5 пунктов из задания.

1) Настроить очередь (Для *ActiveMQ* или любого другого брокера сообщений *JMS*) приема сообщений для администратора.

### 2.1 Добавление зависимостей для работы с брокером ActiveMQ в pom.xml

<dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter-activemq</artifactId>

        </dependency>

        <dependency>

            <groupId>org.apache.activemq</groupId>

            <artifactId>activemq-broker</artifactId>

        </dependency>

### 2.2 Настройка очередей

# Очереди для операций с парфюмами

parfume.queue.getAll=parfume.queue.getAll

parfume.queue.getById=parfume.queue.getById

parfume.queue.add=parfume.queue.add

parfume.queue.update=parfume.queue.update

parfume.queue.delete=parfume.queue.delete

parfume.queue.searchByType=parfume.queue.searchByType

parfume.queue.response=parfume.queue.response

parfume.queue.buy=parfume.queue.buy

parfume.queue.response=your.target.queue.name

### 2.3 Добавление классов для работы с брокером

Класс JmsConfig содержит 2 метода.

Первый метод создает и настраивает фабрику контейнеров слушателей JMS, используя переданный объект ConnectionFactory и конфигуратор DefaultJmsListenerContainerFactoryConfigurer, что позволяет управлять созданием слушателей сообщений. Второй метод определяет конвертер сообщений MappingJackson2MessageConverter, который позволяет преобразовывать объекты Java в текстовые сообщения в формате JSON, устанавливая тип сообщения и имя свойства для идентификации типов данных. Эта конфигурация упрощает работу с сообщениями в приложении, обеспечивая возможность обмена данными между различными компонентами через очередь сообщений.

// Класс конфигурации для настройки JMS (Java Message Service)

@Configuration

public class JmsConfig {

    // Метод для создания фабрики слушателей JMS

    @Bean

    public JmsListenerContainerFactory<?> jmsListenerContainerFactory(

            ConnectionFactory connectionFactory,

            DefaultJmsListenerContainerFactoryConfigurer configurer) {

        // Создаем экземпляр фабрики слушателей

        DefaultJmsListenerContainerFactory factory = new DefaultJmsListenerContainerFactory();

        // Настраиваем фабрику с использованием предоставленной конфигурации и ConnectionFactory

        configurer.configure(factory, connectionFactory);

        // Возвращаем настроенную фабрику слушателей

        return factory;

    }

    // Метод для настройки конвертера сообщений для работы с JSON

    @Bean

    public MessageConverter jacksonJmsMessageConverter() {

        // Создаем экземпляр конвертера, который будет конвертировать сообщения в формат JSON и обратно

        MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();

        // Устанавливаем тип сообщения в текстовый (TEXT)

        converter.setTargetType(MessageType.TEXT);

        // Устанавливаем название свойства, которое будет хранить идентификатор типа сообщения

        converter.setTypeIdPropertyName("\_type");

        // Возвращаем настроенный конвертер сообщений

        return converter;

    }

}

Класс ParfumeResponse представляет собой DTO (Data Transfer Object), который используется для передачи информации о результате обработки запросов, связанных с парфюмерией. Он включает в себя несколько полей, таких как success (для указания успешности операции), message (для передачи сообщения об операции), а также списки и отдельные экземпляры класса Parfume.

// Класс ParfumeResponse реализует интерфейс Serializable

public class ParfumeResponse implements Serializable {

    // Поле, указывающее на успешность операции

    private boolean success;

    // Поле для хранения сообщения, которое описывает результат операции

    private String message;

    // Поле, содержащее список парфюмерии (Parfume)

    private List<Parfume> parfumes;

    // Поле для хранения одного парфюма

    private Parfume parfume;

    // Конструктор без параметров

    public ParfumeResponse() {}

    // Конструктор с параметрами, позволяющий задать успех и сообщение

    public ParfumeResponse(boolean success, String message) {

        this.success = success; // Устанавливаем значение поля success

        this.message = message; // Устанавливаем значение поля message

        this.parfumes = Collections.emptyList(); // Инициализируем список parfumes пустым списком

    }

    // Конструктор, позволяющий задать успех, сообщение и один парфюм

    public ParfumeResponse(boolean success, String message, Parfume parfume) {

        this.success = success; // Устанавливаем значение поля success

        this.message = message; // Устанавливаем значение поля message

        this.parfume = parfume; // Устанавливаем переданный парфюм

    }

    // Конструктор, позволяющий задать успех, сообщение и список парфюмерии

    public ParfumeResponse(boolean success, String message, List<Parfume> parfumes) {

        this.success = success; // Устанавливаем значение поля success

        this.message = message; // Устанавливаем значение поля message

        this.parfumes = parfumes; // Устанавливаем переданный список парфюмерии

    }

    // Геттер для поля success

    public boolean isSuccess() {

        return success; // Возвращаем текущее значение поля success

    }

    // Сеттер для поля success

    public void setSuccess(boolean success) {

        this.success = success; // Устанавливаем значение поля success

    }

    // Геттер для поля message

    public String getMessage() {

        return message; // Возвращаем текущее значение поля message

    }

    // Сеттер для поля message

    public void setMessage(String message) {

        this.message = message; // Устанавливаем значение поля message

    }

    // Геттер для поля parfumes

    public List<Parfume> getParfumes() {

        return parfumes; // Возвращаем список парфюмерии

    }

    // Сеттер для поля parfumes

    public void setParfumes(List<Parfume> parfumes) {

        this.parfumes = parfumes; // Устанавливаем значение поля parfumes

    }

    // Геттер для поля parfume

    public Parfume getParfume() {

        return parfume; // Возвращаем текущий парфюм

    }

    // Сеттер для поля parfume

    public void setParfume(Parfume parfume) {

        this.parfume = parfume; // Устанавливаем значение поля parfume

    }

}

Класс ParfumeMessage представляет собой структуру, предназначенную для обмена сообщениями в приложении, связанном с парфюмерией. Этот класс включает в себя несколько ключевых полей и методов, которые позволяют эффективно управлять информацией о парфюмах.

// Класс ParfumeMessage, реализующий интерфейс Serializable для передачи сообщений

public class ParfumeMessage implements Serializable {

    // Уникальный идентификатор сообщения

    private Long id;

    // Экземпляр класса Parfume, представляющий парфюм

    private Parfume parfume;

    // Тип поиска, который может быть использован для фильтрации запросов

    private String searchType;

    // Конструкторы класса ParfumeMessage

    // Конструктор по умолчанию, который необходим для сериализации

    public ParfumeMessage() {}

    // Конструктор, принимающий идентификатор парфюма

    public ParfumeMessage(Long id) {

        this.id = id; // Устанавливаем переданный идентификатор

    }

    // Конструктор, принимающий объект Parfume

    public ParfumeMessage(Parfume parfume) {

        this.parfume = parfume; // Устанавливаем переданный объект Parfume

    }

    // Конструктор, принимающий тип поиска

    public ParfumeMessage(String searchType) {

        this.searchType = searchType; // Устанавливаем переданный тип поиска

    }

    // Геттеры и сеттеры для доступа к полям

    // Метод для получения идентификатора сообщения

    public Long getId() {

        return id; // Возвращаем уникальный идентификатор

    }

    // Метод для установки идентификатора сообщения

    public void setId(Long id) {

        this.id = id; // Устанавливаем уникальный идентификатор

    }

    // Метод для получения объекта Parfume

    public Parfume getParfume() {

        return parfume; // Возвращаем объект Parfume

    }

    // Метод для установки объекта Parfume

    public void setParfume(Parfume parfume) {

        this.parfume = parfume; // Устанавливаем объект Parfume

    }

    // Метод для получения типа поиска

    public String getSearchType() {

        return searchType; // Возвращаем тип поиска

    }

    // Метод для установки типа поиска

    public void setSearchType(String searchType) {

        this.searchType = searchType; // Устанавливаем тип поиска

    }

}

Класс ParfumeJmsProducer предназначен для работы с системой обмена сообщениями (JMS) в контексте управления данными о парфюмах. Он служит в качестве продюсера сообщений, используя JmsTemplate для отправки различных запросов на обработку парфюмерной информации в очереди.

Основная функция этого класса заключается в отправке сообщений о выполнении различных операций с парфюмами, таких как добавление, обновление, удаление и получение парфюмов, а также обработка запросов на покупку. Класс также предоставляет методы для поиска парфюмов по типу и для получения всех парфюмов.

@Component // Обозначение данного класса как Spring-компонента

public class ParfumeJmsProducer {

    @Autowired // Автоматическое внедрение зависимости JmsTemplate

    private JmsTemplate jmsTemplate;

    // Извлечение имен очередей из файла конфигурации

    @Value("${parfume.queue.getAll}")

    private String getAllQueueName;

    @Value("${parfume.queue.getById}")

    private String getByIdQueueName;

    @Value("${parfume.queue.add}")

    private String addQueueName;

    @Value("${parfume.queue.update}")

    private String updateQueueName;

    @Value("${parfume.queue.delete}")

    private String deleteQueueName;

    @Value("${parfume.queue.searchByType}")

    private String searchByTypeQueueName;

    @Value("${parfume.queue.response}")

    private String responseQueueName;

    @Value("${parfume.queue.buy}")

    private String responseBuy;

    /\*\*

     \* Отправляет данные о покупке парфюма в JMS-очередь.

     \* @param id ID парфюма

     \* @param parfume полные данные о парфюме

     \*/

    public void purchaseParfume(Long id, Parfume parfume) {

        // Создание карты для хранения информации о покупке

        Map<String, Object> purchaseInfo = new HashMap<>();

        purchaseInfo.put("action", "PURCHASE"); // Указываем действие

        purchaseInfo.put("parfumeId", id); // Добавляем ID парфюма

        purchaseInfo.put("parfumeData", parfume); // Добавляем данные парфюма

        purchaseInfo.put("purchaseTime", new Date()); // Записываем текущее время покупки

        // Отправляем сообщение в очередь для покупок

        jmsTemplate.convertAndSend(responseBuy, purchaseInfo);

    }

    /\*\*

     \* Запрос всех парфюмов, отправляя сообщение в соответствующую очередь.

     \*/

    public void getAllParfumes() {

        jmsTemplate.convertAndSend(getAllQueueName, new ParfumeMessage());

    }

    /\*\*

     \* Запрос парфюма по его ID.

     \* @param id ID парфюма, который нужно получить

     \*/

    public void getParfumeById(Long id) {

        ParfumeMessage message = new ParfumeMessage(id); // Создаем сообщение с ID

        jmsTemplate.convertAndSend(getByIdQueueName, message); // Отправляем сообщение в очередь

    }

    /\*\*

     \* Добавляет новый парфюм, отправляя его данные в очередь.

     \* @param parfume объект Parfume, который нужно добавить

     \*/

    public void addParfume(Parfume parfume) {

        ParfumeMessage message = new ParfumeMessage(parfume); // Создаем сообщение с данными парфюма

        jmsTemplate.convertAndSend(addQueueName, message); // Отправляем сообщение в очередь добавления

    }

    /\*\*

     \* Обновляет данные о парфюме, отправляя сообщение в соответствующую очередь.

     \* @param parfume объект Parfume с обновленными данными

     \*/

    public void updateParfume(Parfume parfume) {

        ParfumeMessage message = new ParfumeMessage(parfume); // Создаем сообщение с обновленными данными

        jmsTemplate.convertAndSend(updateQueueName, message); // Отправляем сообщение в очередь обновления

    }

    /\*\*

     \* Удаляет парфюм по его ID, отправляя сообщение в очередь на удаление.

     \* @param id ID парфюма, который нужно удалить

     \*/

    public void deleteParfume(Long id) {

        ParfumeMessage message = new ParfumeMessage(id); // Создаем сообщение с ID парфюма

        jmsTemplate.convertAndSend(deleteQueueName, message); // Отправляем сообщение в очередь удаления

    }

    /\*\*

     \* Выполняет поиск парфюмов по типу, отправляя сообщение с типом поиска.

     \* @param type тип парфюма для поиска

     \*/

    public void searchParfumesByType(String type) {

        ParfumeMessage message = new ParfumeMessage(); // Создаем сообщение

        message.setSearchType(type); // Устанавливаем тип поиска

        jmsTemplate.convertAndSend(searchByTypeQueueName, message); // Отправляем сообщение в очередь поиска

    }

}

Класс ParfumeJmsListener включает методы, которые позволяют выполнять операции получения, добавления, обновления и удаления парфюмов. Каждый метод слушает определённые очереди и в ответ на запросы возвращает результаты в виде объекта ParfumeResponse, сообщая пользователю о статусе выполнения операций или о возникших ошибках.

package ru.example.JMS;

// Импортирование необходимых классов из Spring и других пакетов

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

import org.springframework.jms.annotation.JmsListener;

import org.springframework.messaging.handler.annotation.SendTo;

import org.springframework.stereotype.Component;

import ru.example.Parfume; // Импорт класса парфюма

import ru.example.ParfumeService; // Импорт сервиса, который управляет парфюмами

import java.util.List; // Импорт класса для работы со списками

import java.util.Map; // Импорт класса для работы с картами (ассоциативные массивы)

// Указывает, что этот класс является компонентом Spring и должен быть включён в контекст приложения

@Component

public class ParfumeJmsListener {

    // Автоматическая реализация зависимости ParfumeService

    @Autowired

    private ParfumeService parfumeService;

    // Обработчик запросов на получение всех парфюмов из очереди, адрес которой задаётся в конфигурации

    @JmsListener(destination = "${parfume.queue.getAll}")

    @SendTo("${parfume.queue.response}") // Отправлять ответ в заданную очередь

    public ParfumeResponse handleGetAllRequest(ParfumeMessage message) {

        try {

            // Получение списка всех парфюмов из сервиса

            List<Parfume> parfumes = parfumeService.getAllParfume();

            // Формирование успешного ответа с полученным списком парфюмов

            return new ParfumeResponse(true, "Успешно получен список парфюмов", parfumes);

        } catch (Exception e) {

            // Обработка ошибок и возврат ответа с сообщением об ошибке

            return new ParfumeResponse(false, "Ошибка при получении списка парфюмов: " + e.getMessage());

        }

    }

    // Обработчик запросов на получение парфюма по ID

    @JmsListener(destination = "${parfume.queue.getById}")

    @SendTo("${parfume.queue.response}")

    public ParfumeResponse handleGetByIdRequest(ParfumeMessage message) {

        try {

            // Получение парфюма по ID, переданному в сообщении

            Parfume parfume = parfumeService.getParfumeById(message.getId());

            // Проверка, найден ли парфюм

            if (parfume != null) {

                return new ParfumeResponse(true, "Парфюм успешно найден", parfume);

            } else {

                return new ParfumeResponse(false, "Парфюм с ID " + message.getId() + " не найден");

            }

        } catch (Exception e) {

            return new ParfumeResponse(false, "Ошибка при получении парфюма: " + e.getMessage());

        }

    }

    // Обработчик запросов на добавление нового парфюма

    @JmsListener(destination = "${parfume.queue.add}")

    @SendTo("${parfume.queue.response}")

    public ParfumeResponse handleAddRequest(ParfumeMessage message) {

        try {

            // Сохранение нового парфюма в базе данных

            Parfume savedParfume = parfumeService.saveParfume(message.getParfume());

            return new ParfumeResponse(true, "Парфюм успешно добавлен", savedParfume);

        } catch (Exception e) {

            return new ParfumeResponse(false, "Ошибка при добавлении парфюма: " + e.getMessage());

        }

    }

    // Обработчик запросов на обновление существующего парфюма

    @JmsListener(destination = "${parfume.queue.update}")

    @SendTo("${parfume.queue.response}")

    public ParfumeResponse handleUpdateRequest(ParfumeMessage message) {

        try {

            // Получение объекта парфюма из сообщения

            Parfume parfume = message.getParfume();

            // Обновление парфюма и возврат ответом обновленного объекта

            Parfume updatedParfume = parfumeService.updateParfume(parfume.getId(), parfume);

            return new ParfumeResponse(true, "Парфюм успешно обновлен", updatedParfume);

        } catch (IllegalArgumentException e) {

            return new ParfumeResponse(false, "Парфюм с указанным ID не найден: " + e.getMessage());

        } catch (Exception e) {

            return new ParfumeResponse(false, "Ошибка при обновлении парфюма: " + e.getMessage());

        }

    }

    // Обработчик запросов на удаление парфюма по ID

    @JmsListener(destination = "${parfume.queue.delete}")

    @SendTo("${parfume.queue.response}")

    public ParfumeResponse handleDeleteRequest(ParfumeMessage message) {

        try {

            Long id = message.getId();

            // Удаление парфюма и возврат результата удаления

            boolean deleted = parfumeService.deleteParfume(id);

            if (deleted) {

                return new ParfumeResponse(true, "Парфюм успешно удален");

            } else {

                return new ParfumeResponse(false, "Парфюм с ID " + id + " не найден");

            }

        } catch (Exception e) {

            return new ParfumeResponse(false, "Ошибка при удалении парфюма: " + e.getMessage());

        }

    }

    // Обработчик запросов на поиск парфюма по типу

    @JmsListener(destination = "${parfume.queue.searchByType}")

    @SendTo("${parfume.queue.response}")

    public ParfumeResponse handleSearchByTypeRequest(ParfumeMessage message) {

        try {

            String type = message.getSearchType();

            // Поиск парфюмов по типу в сервисе

            List<Parfume> parfumes = parfumeService.searchByType(type);

            return new ParfumeResponse(true, "Парфюмы успешно найдены по типу: " + type, parfumes);

        } catch (Exception e) {

            return new ParfumeResponse(false, "Ошибка при поиске парфюмов по типу: " + e.getMessage());

        }

    }

    /\*\*

     \* Обрабатывает запрос на покупку парфюма.

     \* @param purchaseInfo информация о покупке

     \* @return результат обработки покупки

     \*/

    @JmsListener(destination = "${parfume.queue.buy}")

    @SendTo("${parfume.queue.response}")

    public ParfumeResponse handlePurchaseRequest(Map<String, Object> purchaseInfo) {

        try {

            // Извлекаем данные о парфюме из карты

            Long parfumeId = Long.valueOf(purchaseInfo.get("parfumeId").toString());

            // Получаем парфюм по ID из базы данных

            Parfume parfume = parfumeService.getParfumeById(parfumeId);

            // Проверка, найден ли парфюм

            if (parfume == null) {

                return new ParfumeResponse(false, "Парфюм с ID " + parfumeId + " не найден");

            }

            // Здесь можно добавить логику обработки покупки, например, уменьшение количества на складе

            // parfumeService.processPurchase(parfumeId);

            return new ParfumeResponse(true, "Покупка парфюма успешно обработана", parfume);

        } catch (Exception e) {

            return new ParfumeResponse(false, "Ошибка при обработке покупки парфюма: " + e.getMessage());

        }

    }

}

2) При выполнении операций добавления, удаления или редактирования ресурса через *REST API* / форму создавать соответствующие уведомления и отправлять их в очередь.

### 2.4 Внесение изменений в класс ParfumeRestController

/\*\*

 \* REST контроллер для работы с парфюмами.

 \* Поддерживает как синхронную обработку через ParfumeService,

 \* так и асинхронную обработку через JMS (ParfumeJmsProducer).

 \* При любой операции отправляет JMS-сообщение в очередь.

 \*/

@RestController

@RequestMapping("/api/parfume")  // Устанавливаем базовый путь для всех методов контроллера

public class ParfumeRestController {

    @Autowired

    private ParfumeService parfumeService;  // Сервис для работы с парфюмами

    @Autowired

    private ParfumeJmsProducer jmsProducer;  // Производитель JMS-сообщений для асинхронной обработки

    /\*\*

     \* Получение всех парфюмов.

     \* @param async Если true, возвращает только статус отправки запроса в JMS

     \* @return ResponseEntity со списком парфюмов или статусом запроса

     \*/

    @GetMapping

    public ResponseEntity<?> getAllParfumes(@RequestParam(value = "async", defaultValue = "false") boolean async) {

        // Всегда отправляем сообщение в JMS для получения всех парфюмов

        jmsProducer.getAllParfumes();

        sendAndCheck();  // Проверка состояния очереди после отправки

        if (async) {

            // Если запрашивается асинхронный режим, возвращаем статус запроса

            return ResponseEntity.accepted().body("Запрос на получение всех парфюмов отправлен");

        } else {

            // В синхронном режиме выполняем запрос напрямую к сервису

            List<Parfume> parfumes = parfumeService.getAllParfume();

            return new ResponseEntity<>(parfumes, HttpStatus.OK);  // Возвращаем список парфюмов с успешным статусом

        }

    }

    /\*\*

     \* Получение парфюма по ID.

     \* @param id ID парфюма

     \* @param async Если true, возвращает только статус отправки запроса в JMS

     \* @return ResponseEntity с парфюмом или статусом запроса

     \*/

    @GetMapping("/{id}")

    public ResponseEntity<?> getParfumeById(

            @PathVariable Long id,  // Извлекаем ID из URL

            @RequestParam(value = "async", defaultValue = "false") boolean async) {

        // Отправляем сообщение в JMS для получения парфюма по ID

        jmsProducer.getParfumeById(id);

        sendAndCheck();  // Проверка состояния очереди

        if (async) {

            // Для асинхронного режима возвращаем статус отправки запроса

            return ResponseEntity.accepted().body("Запрос на получение парфюма с ID " + id + " отправлен");

        } else {

            // Для синхронного запроса получаем парфюм по ID

            Parfume parfume = parfumeService.getParfumeById(id);

            // Проверяем, существует ли парфюм

            if (parfume != null) {

                return new ResponseEntity<>(parfume, HttpStatus.OK);  // Успешный ответ с парфюмом

            } else {

                return new ResponseEntity<>(HttpStatus.NOT\_FOUND);  // Возвращаем статус 404, если парфюм не найден

            }

        }

    }

    /\*\*

     \* Создание нового парфюма.

     \* @param parfume Данные парфюма

     \* @param async Если true, возвращает только статус отправки запроса в JMS

     \* @return ResponseEntity с созданным парфюмом или статусом запроса

     \*/

    @PostMapping

    public ResponseEntity<?> createParfume(

            @RequestBody Parfume parfume,  // Данные парфюма передаются в теле запроса

            @RequestParam(value = "async", defaultValue = "false") boolean async) {

        // Отправляем сообщение в JMS для создания нового парфюма

        if (async) {

            // Если запрашивается асинхронный режим, возвращаем статус отправки запроса

            return ResponseEntity.accepted().body("Запрос на добавление парфюма отправлен");

        } else {

            try {

                jmsProducer.addParfume(parfume);  // Отправка парфюма в JMS

                sendAndCheck();  // Проверка состояния очереди

                return new ResponseEntity<>(HttpStatus.CREATED);  // Успешный ответ с кодом 201

            } catch (Exception e) {

                return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);  // Обработка ошибок

            }

        }

    }

    /\*\*

     \* Обновление существующего парфюма.

     \* @param id ID парфюма

     \* @param parfumeDetails Новые данные парфюма

     \* @param async Если true, возвращает только статус отправки запроса в JMS

     \* @return ResponseEntity с обновленным парфюмом или статусом запроса

     \*/

    @PutMapping("/{id}")

    public ResponseEntity<?> updateParfume(

            @PathVariable Long id,  // Извлечение ID из URL

            @RequestBody Parfume parfumeDetails,  // Новые данные парфюма

            @RequestParam(value = "async", defaultValue = "false") boolean async) {

        // Установка ID парфюма для обновления

        parfumeDetails.setId(id);

        // Отправляем сообщение в JMS для обновления парфюма

        jmsProducer.updateParfume(parfumeDetails);

        sendAndCheck();  // Проверка состояния очереди

        if (async) {

            // Если запрашивается асинхронный режим, возвращаем статус отправки запроса

            return ResponseEntity.accepted().body("Запрос на обновление парфюма с ID " + id + " отправлен");

        } else {

            try {

                // Обновление парфюма через сервис

                Parfume updatedParfume = parfumeService.updateParfume(id, parfumeDetails);

                return new ResponseEntity<>(updatedParfume, HttpStatus.OK);  // Возвращаем обновленный парфюм

            } catch (IllegalArgumentException e) {

                return new ResponseEntity<>(HttpStatus.NOT\_FOUND);  // Парфюм не найден

            } catch (Exception e) {

                return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);  // Обработка ошибок

            }

        }

    }

    /\*\*

     \* Удаление парфюма.

     \* @param id ID парфюма

     \* @param async Если true, возвращает только статус отправки запроса в JMS

     \* @return ResponseEntity со статусом операции

     \*/

    @DeleteMapping("/{id}")

    public ResponseEntity<?> deleteParfume(

            @PathVariable Long id,  // Извлечение ID из URL

            @RequestParam(value = "async", defaultValue = "false") boolean async) {

        // Отправляем сообщение в JMS для удаления парфюма

        jmsProducer.deleteParfume(id);

        sendAndCheck();  // Проверка состояния очереди

        if (async) {

            // Если запрашивается асинхронный режим, возвращаем статус отправки запроса

            return ResponseEntity.accepted().body("Запрос на удаление парфюма с ID " + id + " отправлен");

        } else {

            try {

                boolean deleted = parfumeService.deleteParfume(id);  // Удаление парфюма через сервис

                if (deleted) {

                    return new ResponseEntity<>(HttpStatus.NO\_CONTENT);  // Успешный ответ с кодом 204

                } else {

                    return new ResponseEntity<>(HttpStatus.NOT\_FOUND);  // Парфюм не найден

                }

            } catch (Exception e) {

                return new ResponseEntity<>(HttpStatus.INTERNAL\_SERVER\_ERROR);  // Обработка ошибок

            }

        }

    }

    /\*\*

     \* Поиск парфюмов по типу.

     \* @param type Тип парфюма для поиска

     \* @param async Если true, возвращает только статус отправки запроса в JMS

     \* @return ResponseEntity со списком найденных парфюмов или статусом запроса

     \*/

    @GetMapping("/search")

    public ResponseEntity<?> searchParfumesByType(

            @RequestParam(value = "type") String type,  // Тип парфюма передается в параметрах запроса

            @RequestParam(value = "async", defaultValue = "false") boolean async) {

        // Проверка на наличие типа парфюма

        if (type == null || type.isEmpty()) {

            return new ResponseEntity<>(HttpStatus.BAD\_REQUEST);  // 400 Bad Request если тип не указан

        }

        // Отправляем сообщение в JMS для поиска по типу

        jmsProducer.searchParfumesByType(type);

        sendAndCheck();  // Проверка состояния очереди

        if (async) {

            // Если запрашивается асинхронный режим, возвращаем статус отправки запроса

            return ResponseEntity.accepted().body("Запрос на поиск парфюмов по типу '" + type + "' отправлен");

        } else {

            // Выполнение поиска через сервис

            List<Parfume> parfumes = parfumeService.searchByType(type);

            return new ResponseEntity<>(parfumes, HttpStatus.OK);  // Возвращаем список найденных парфюмов

        }

    }

    /\*\*

     \* Обработка покупки парфюма.

     \* @param id ID парфюма

     \* @return ResponseEntity со статусом операции

     \*/

    @PostMapping("/{id}/purchase")

    public ResponseEntity<?> purchaseParfume(@PathVariable Long id) {

        try {

            // Получаем парфюм по ID для подтверждения покупки

            Parfume parfume = parfumeService.getParfumeById(id);

            if (parfume == null) {

                return new ResponseEntity<>("Парфюм с ID " + id + " не найден", HttpStatus.NOT\_FOUND);  // 404 если парфюм не найден

            }

            // Сохраняем название парфюма для сообщения

            String parfumeName = parfume.getName();

            // Отправляем данные о покупке в JMS-очередь

            jmsProducer.purchaseParfume(id, parfume);

            sendAndCheck();  // Проверка состояния очереди

            // Удаляем парфюм из базы данных сразу после отправки сообщения

            boolean deleted = parfumeService.deleteParfume(id);

            if (deleted) {

                return ResponseEntity.ok("Парфюм '" + parfumeName + "' успешно куплен и удален из базы данных");  // Успешный ответ

            } else {

                return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)

                        .body("Покупка оформлена, но возникла ошибка при удалении парфюма из базы данных");  // Ошибка при удалении

            }

        } catch (Exception e) {

            return new ResponseEntity<>("Ошибка при оформлении заказа: " + e.getMessage(),

                    HttpStatus.INTERNAL\_SERVER\_ERROR);  // Обработка ошибок

        }

    }

    // Пример кода для проверки после отправки сообщения

    public void sendAndCheck() {

        System.out.println("Сообщение отправлено, проверяем очередь:");

        MessageBrowser.main(new String[]{});  // Вызов метода для проверки состояния очереди

        // Добавляем небольшую задержку, чтобы увидеть сообщение в очереди

        try {

            Thread.sleep(1000); // Ждем 1 секунду

        } catch (InterruptedException e) {

            Thread.currentThread().interrupt();  // Восстанавливаем состояние потока

        }

    }

}

3) Любым удобным способом (можно через терминал) продемонстрировать извлечение административных сообщений о выполненных операциях (из п.2).

Метод для отображения сообщений об операциях в консоли:

/\*\*

     \* JMS слушатель для получения ответов асинхронно

     \* Этот метод будет вызван Spring JMS, когда сообщение придет в очередь ответов

     \*/

    @JmsListener(destination = "${parfume.queue.response}")

    public void receiveJmsResponse(ParfumeResponse response) {

        System.out.println("Получен ответ: " + (response.isSuccess() ? "УСПЕШНО" : "ОШИБКА"));

        System.out.println("Сообщение: " + response.getMessage());

        if (responsePromise != null && !responsePromise.isDone()) {

            responsePromise.complete(response);

        } else {

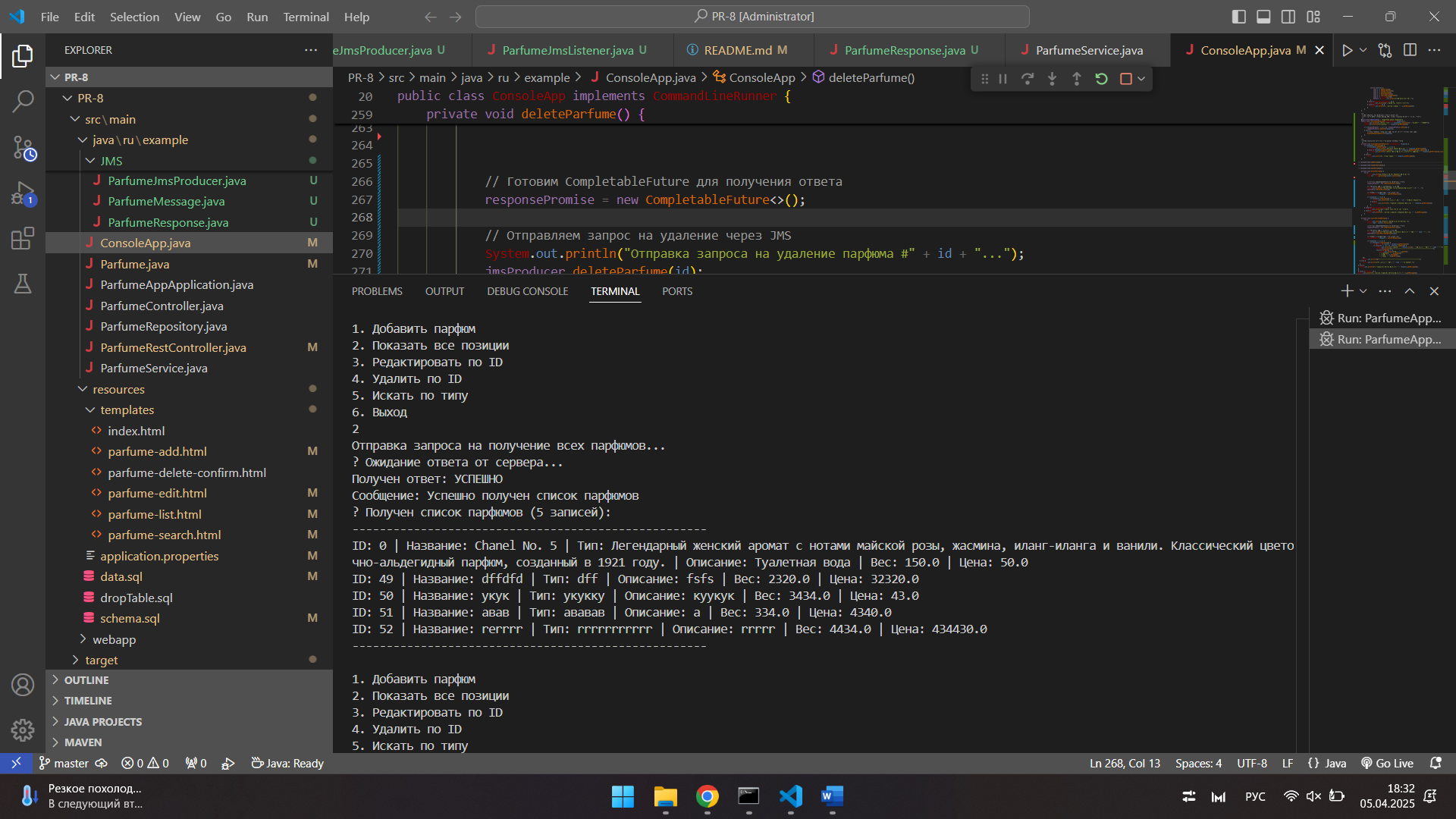
            // Если текущего ожидающего запроса нет, просто выводим информацию

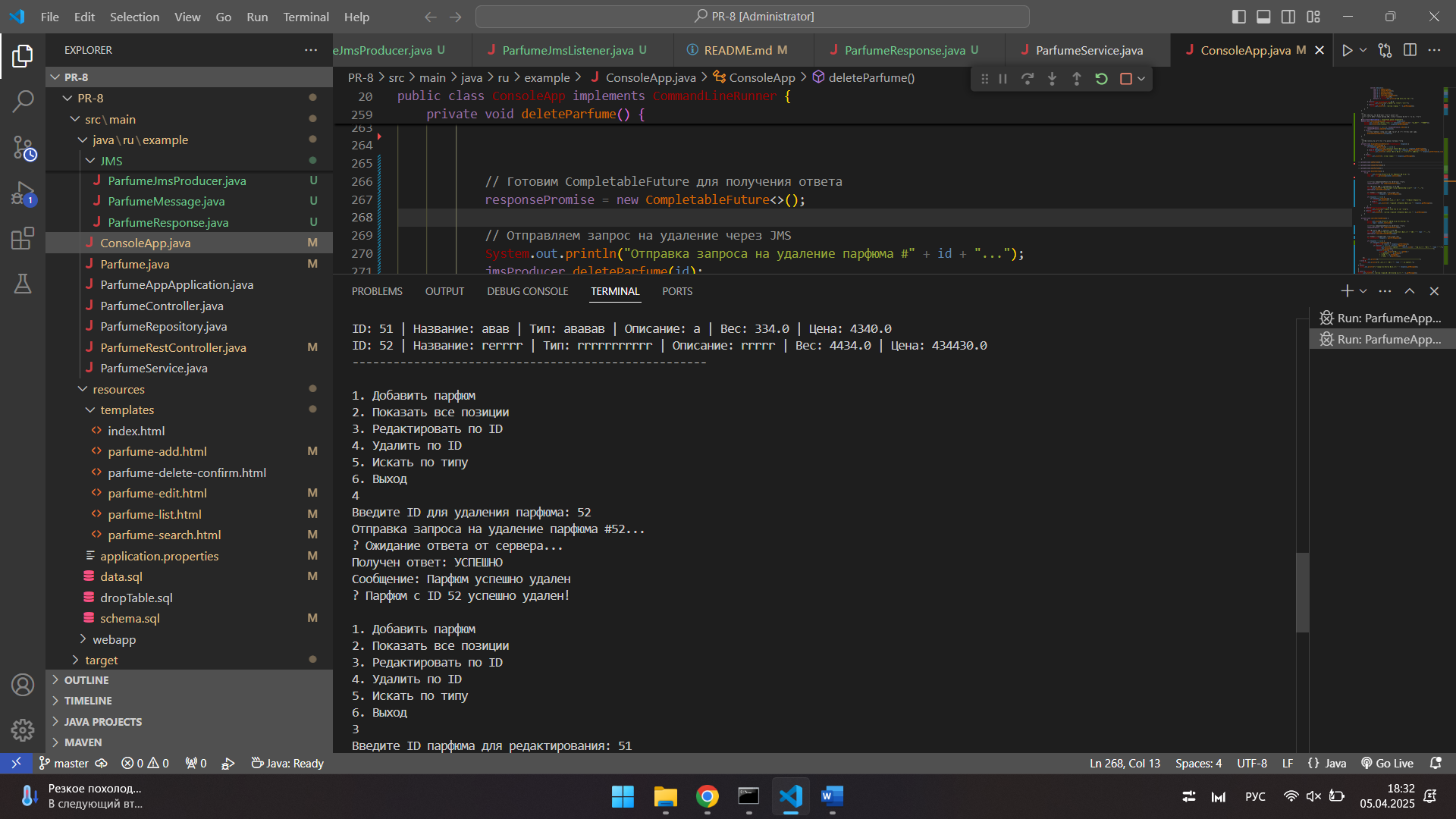
            displayResponseDetails(response);

        }

    }

Пример вывода сообщений в консоль:





4) Добавить кнопку-ссылку «купить» на форме. После этого в брокер сообщений отправляется сообщение о том, какой «товар»/сущность хочет купить пользователь.

5\*) В п.4 "товар" помечается как купленный и не будет показан в общем списке товаров. Необходимо добавить соответствующий столбец, или просто удалить запись о купленном товаре из БД, но перед этим не забыть отправить информацию о товаре в брокер сообщений.

Поменяем макет страницы со списком Парфюма:

<!DOCTYPE html>

<html lang="ru" xmlns:th="http://www.thymeleaf.org">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Список всего парфюма</title>

    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>

</head>

<body>

    <h1>Магазин парфюмерии</h1>

    <ul>

        <li><a href="/parfume/new">1. Добавить парфюм</a></li>

        <li><a href="/parfume">2. Просмотр всего парфюма</a></li>

        <li><a href="/parfume/search">3. Поиск парфюма по типу</a></li>

        <li><a href="/parfume/exit">4. Выход</a></li>

    </ul>

    <h2>Список парфюма</h2>

    <table>

        <thead>

            <tr>

                <th>id</th>

                <th>Название</th>

                <th>Тип</th>

                <th>Описание</th>

                <th>Вес</th>

                <th>Цена</th>

                <th>Действия</th>

            </tr>

        </thead>

        <tbody>

            <tr th:each="parfume : ${parfumes}" th:id="'parfume-row-' + ${parfume.id}">

                <td th:text="${parfume.id}">id</td>

                <td th:text="${parfume.name}">Название</td>

                <td th:text="${parfume.type}">Тип</td>

                <td th:text="${parfume.description}">Описание</td>

                <td th:text="${parfume.weight}">Вес</td>

                <td th:text="${parfume.price}">Цена</td>

                <td>

                    <button th:onclick="'purchaseParfume(' + ${parfume.id} + ')'">Купить</button>

                    <a th:href="@{/parfume/edit/{id}(id=${parfume.id})}">Редактировать</a>

                    <a th:onclick="'deleteParfume(' + ${parfume.id} + ')'">Удалить</a>

                </td>

            </tr>

        </tbody>

    </table>

    <a href="/">Закрыть список парфюма</a>

    <script>

        function purchaseParfume(id) {

            if (confirm('Вы уверены, что хотите купить этот парфюм?')) {

                $.ajax({

                    url: '/api/parfume/' + id + '/purchase',

                    type: 'POST',

                    success: function(response) {

                        // Показываем уведомление о успешной покупке

                        alert(response);

                        // Удаляем строку из таблицы без перезагрузки страницы

                        $('#parfume-row-' + id).fadeOut(500, function() {

                            $(this).remove();

                        });

                    },

                    error: function(xhr) {

                        alert('Ошибка при оформлении заказа: ' + xhr.responseText);

                    }

                });

            }

        }

        function deleteParfume(id) {

            if (confirm('Вы уверены, что хотите удалить этот парфюм?')) {

                $.ajax({

                    url: '/api/parfume/' + id,

                    type: 'DELETE',

                    success: function() {

                        alert('Парфюм успешно удален');

                        // Удаляем строку из таблицы без перезагрузки страницы

                        $('#parfume-row-' + id).fadeOut(500, function() {

                            $(this).remove();

                        });

                    },

                    error: function() {

                        alert('Ошибка при удалении парфюма');

                    }

                });

            }

        }

    </script>

</body>

</html>

Добавим в класс ParfumeJmsProducer:

/\*\*

     \* Отправляет данные о покупке парфюма в JMS-очередь.

     \* @param id ID парфюма

     \* @param parfume полные данные о парфюме

     \*/

    public void purchaseParfume(Long id, Parfume parfume) {

        Map<String, Object> purchaseInfo = new HashMap<>();

        purchaseInfo.put("action", "PURCHASE");

        purchaseInfo.put("parfumeId", id);

        purchaseInfo.put("parfumeData", parfume);

        purchaseInfo.put("purchaseTime", new Date());

        // Отправляем сообщение в очередь покупок

        jmsTemplate.convertAndSend(responseBuy, purchaseInfo);

    }

Добавим в класс ParfumeRestController:

    /\*\*

     \* Обработка покупки парфюма.

     \* @param id ID парфюма

     \* @return ResponseEntity со статусом операции

     \*/

    @PostMapping("/{id}/purchase")

    public ResponseEntity<?> purchaseParfume(@PathVariable Long id) {

        try {

            // Получаем парфюм

            Parfume parfume = parfumeService.getParfumeById(id);

            if (parfume == null) {

                return new ResponseEntity<>("Парфюм с ID " + id + " не найден", HttpStatus.NOT\_FOUND);

            }

            // Сохраняем название для сообщения

            String parfumeName = parfume.getName();

            // Отправляем данные о покупке в JMS-очередь

            jmsProducer.purchaseParfume(id, parfume);

            sendAndCheck();

            // Удаляем парфюм из базы данных сразу после отправки сообщения

            boolean deleted = parfumeService.deleteParfume(id);

            if (deleted) {

                return ResponseEntity.ok("Парфюм '" + parfumeName + "' успешно куплен и удален из базы данных");

            } else {

                return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR)

                        .body("Покупка оформлена, но возникла ошибка при удалении парфюма из базы данных");

            }

        } catch (Exception e) {

            return new ResponseEntity<>("Ошибка при оформлении заказа: " + e.getMessage(),

                    HttpStatus.INTERNAL\_SERVER\_ERROR);

        }

    }

Добавим в класс ParfumeJmsListener:

/\*\*

     \* Обрабатывает запрос на покупку парфюма.

     \* @param purchaseInfo информация о покупке

     \* @return результат обработки покупки

     \*/

    @JmsListener(destination = "${parfume.queue.buy}")

    @SendTo("${parfume.queue.response}")

    public ParfumeResponse handlePurchaseRequest(Map<String, Object> purchaseInfo) {

        try {

            // Извлекаем данные из сообщения

            Long parfumeId = Long.valueOf(purchaseInfo.get("parfumeId").toString());

            // Получаем парфюм из базы данных по ID вместо использования переданного объекта

            Parfume parfume = parfumeService.getParfumeById(parfumeId);

            if (parfume == null) {

                return new ParfumeResponse(false, "Парфюм с ID " + parfumeId + " не найден");

            }

            // Здесь добавляем логику обработки покупки

            // parfumeService.processPurchase(parfumeId);

            return new ParfumeResponse(true, "Покупка парфюма успешно обработана", parfume);

        } catch (Exception e) {

            return new ParfumeResponse(false, "Ошибка при обработке покупки парфюма: " + e.getMessage());

        }

    }

Демонстрация работы программы:

Нажатие кнопки «Купить»

