Yêu cầu

Giả sử bạn có một hệ thống đặt hàng từ xa. Client thông qua ứng dụng (desktop/web) gửi thông tin đặt hàng sau khi chọn lựa. Thông tin này được chuyển về dạng json và được mã hóa (có thể dùng public/private keys hoặc mã hóa theo mật khẩu hay 1 loại đơn giản nào đó-Base64)

Để tăng hiệu suất xử lý, hệ thống sử dụng messaging service cho việc lắng nghe việc đặt hàng. Khi nhận được đơn hàng từ client, hệ thống sẽ kiểm tra số lượng trong kho có đủ không (có thể mở rộng cho trường hợp kểm tra dịch vụ giao nhận theo địa chỉ, kiểm tra năng lực thanh toán, …) sau đó quyết định đơn hàng có được xác nhận hay không. Cho dù xác nhận hay hủy bỏ, hệ thống cũng sẽ *gửi một email* đến khách hàng để thông báo tình hình.

Hướng dẫn

1. Tạo 1 dự án sử dụng spring-boot với dependency đến **spring-boot-starter-activemq**. Các depedncies

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-activemq'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'org.mariadb.jdbc:mariadb-java-client'
    annotationProcessor 'org.projectlombok:lombok'
    implementation 'com.google.code.gson:gson:2.10.1'

    //for email sending
    implementation 'org.eclipse.angus:angus-mail:2.0.3'
//    implementation 'jakarta.mail:jakarta.mail-api:2.1.3'

    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

2. Tạo csdl cho việc lưu trữ hàng đơn giản. Có thể bao quát các yêu cầu trong 1 bảng *product* (*với lý do là bài tập đơn giản. Thực tế thì không được làm như thế mà cần nhiều vấn đề liên quan.*)
   Một bảng *product_order* để lưu thông tin đặt hàng (mã kh, ….).
   Yêu cầu thêm các bảng nếu có thể để giải quyết bài toán.
   ** *Dùng JPA tạo các entities và mối quan hệ.*
3. Tải ActiveMQ , cấu hình và chạy broker.
4. Sử dụng spring data để viết các lớp theo *kiến trúc layer* để truy xuất csdl và thực hiện các yêu cầu.

Cấu hình cho activemq – application.properties:

```
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=admin
```

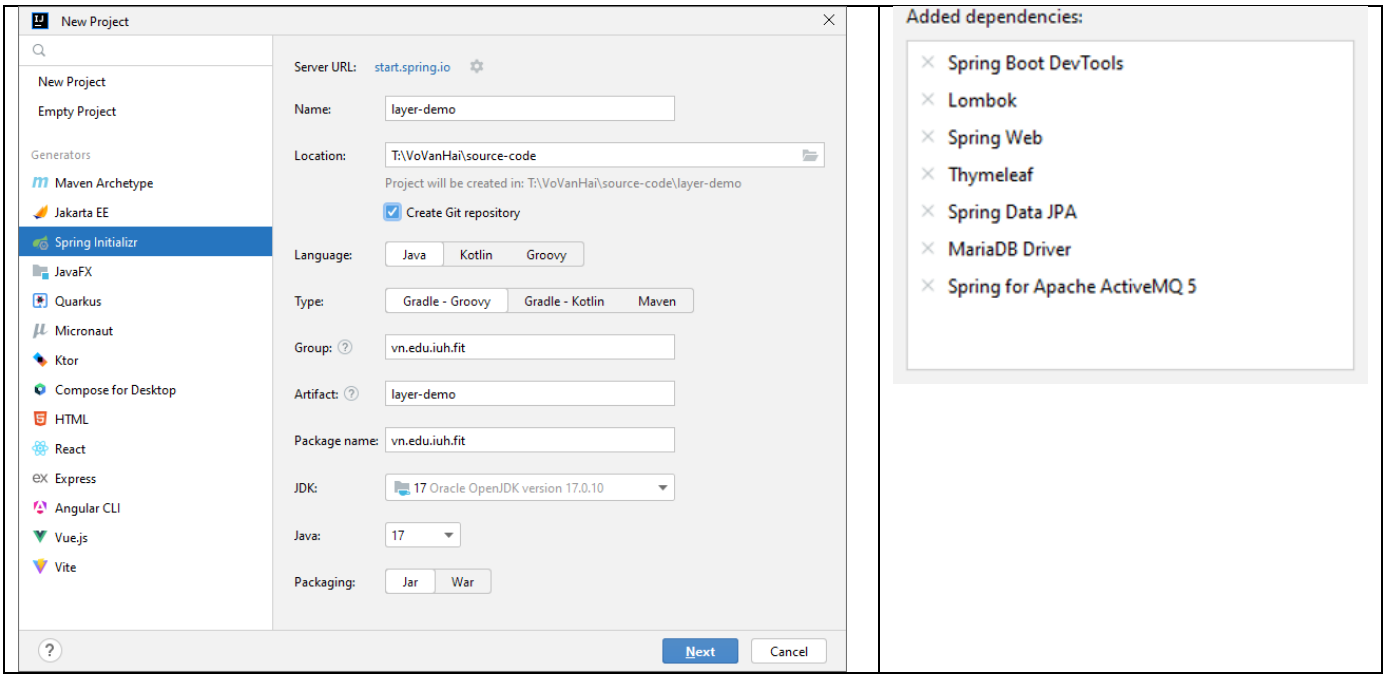
Cấu hình cho database

```
#Database config
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.datasource.url=jdbc:mariadb://localhost:3306/product_ms?createDatabaseIfNotExist=True
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.hibernate.ddl-auto=update
```

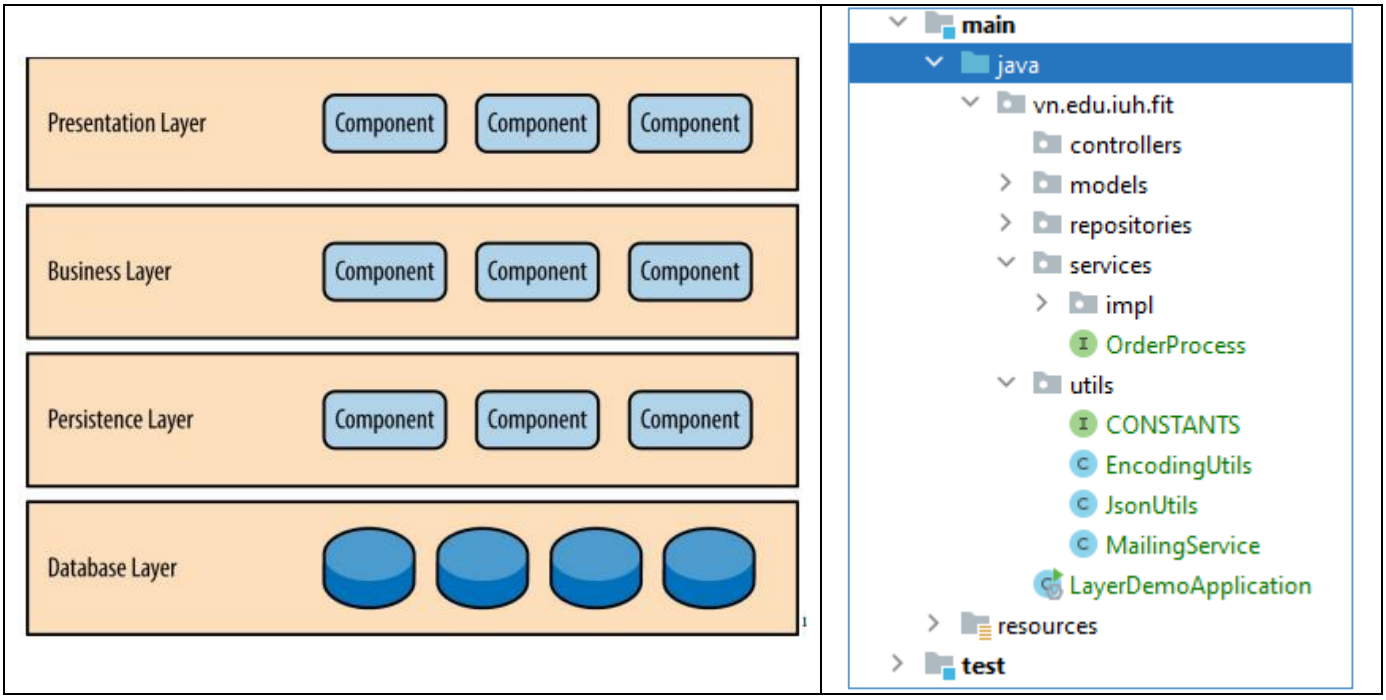
Viết code cho việc lắng nghe trên một queue (name *order_products*)

```
import com.google.gson.Gson;
import org.springframework.jms.annotation.JmsListener;
import org.springframework.messaging.handler.annotation.SendTo;
import org.springframework.stereotype.Component;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.TextMessage;
import java.util.Map;

@Component
public class ProductOrderListener {

    @JmsListener(destination = "order_products")
    public void receiveMessage(final Message jsonMessage) throws JMSException {
        String messageData = null;
        String response = null;
        if(jsonMessage instanceof TextMessage) {
            //1. read message data
            //2. ==> decode
            //3. check for quantity
            //4. make order or reject
            //5. send email
        }
    }
}
```

Tham khảo về ActiveMQ với Springboot:
https://docs.spring.io/spring-boot/docs/3.3.x/reference/html/messaging.html#messaging



Architeccture



Share-services

JsonUtils: Tiện ích chuyển 1 danh dách sản phẩm thành chuỗi Json và ngược lại

```
package vn.edu.iuh.fit.utils;

import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import vn.edu.iuh.fit.models.Product;
```

```
import java.lang.reflect.Type;
import java.util.List;

public class JsonUtils {
    public static String convertListProducts2Json(List<Product> productList) {
        Gson gson = new Gson();
        return gson.toJson(productList);
    }

    private static List<Product> convertJson2ListProduct(String json) {
        Gson gson = new Gson();
        Type type = new TypeToken<List<Product>>() {
        }.getType();
        return gson.fromJson(json, type);
    }
}
```

Crypting pakage định nghĩa các lớp hỗ trợ mã hóa. Trong trường hợp này có lại loại: base64 và password-based

```
package vn.edu.iuh.fit.utils.crypting;

import javax.crypto.SecretKey;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;

public interface EncodingText {
    final String CRYPT_ALGORITHM = "AES";
    final String DEFAULT_PASSWORD = "S3cr3t";
    String encrypt(String plainText)throws Exception;
    public String decrypt(String encryptedText)throws Exception ;
}
```

```
package vn.edu.iuh.fit.utils.crypting;

import org.apache.tomcat.util.codec.binary.Base64;

public class Base64EncodingText implements EncodingText {
    public String encrypt(String jsonText) throws Exception {
        return Base64.encodeBase64String(jsonText.getBytes());
    }

    public String decrypt(String encodeJson) throws Exception {
        return new String(Base64.decodeBase64(encodeJson));
    }
}
```

Để dùng password-based crypting, ta phải cung cấp một secrect-key. Trong lớp cung cấp sẵn phương thức tạo 1 key, lưu key vào file cũng như load key từ file để mã hóa.

```
package vn.edu.iuh.fit.utils.crypting;

import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
```

```
import java.security.Key;
import java.util.Base64;

public class PasswordEncodingText implements EncodingText {
    private SecretKey secretKey = null;

    public SecretKey getSecretKey() {
        return secretKey;
    }

    public void setSecretKey(SecretKey secretKey) {
        this.secretKey = secretKey;
    }

    @Override
    public String encrypt(String plainText) throws Exception {
        if (secretKey == null)
            throw new Exception("Secret key is needed. Load secret key");
        byte[] plainTextByte = plainText.getBytes();
        Cipher cipher = Cipher.getInstance(CRYPT_ALGORITHM);
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedByte = cipher.doFinal(plainTextByte);
        Base64.Encoder encoder = Base64.getEncoder();
        return encoder.encodeToString(encryptedByte);
    }

    public String decrypt(String encryptedText) throws Exception {
        if (secretKey == null)
            throw new Exception("Secret key is needed. Load secret key");
        Base64.Decoder decoder = Base64.getDecoder();
        byte[] encryptedTextByte = decoder.decode(encryptedText);
        Cipher cipher = Cipher.getInstance(CRYPT_ALGORITHM);
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decryptedByte = cipher.doFinal(encryptedTextByte);
        return new String(decryptedByte);
    }

    public void saveKey(String path, Key myKey) throws Exception {
        try (ObjectOutputStream oout = new ObjectOutputStream(new
FileOutputStream(path))) {
            oout.writeObject(myKey);
        }
    }

    public Key loadKey(String path) throws Exception {
        Key key = null;
        try (ObjectInputStream oout = new ObjectInputStream(new
FileInputStream(path))) {
            key = (Key) oout.readObject();
        }
        return key;
    }

    public SecretKey createSecretKey() throws Exception {
        KeyGenerator keyGenerator = KeyGenerator.getInstance(CRYPT_ALGORITHM);
        keyGenerator.init(128); // block size is 128bits
        return keyGenerator.generateKey();
    }
}
```