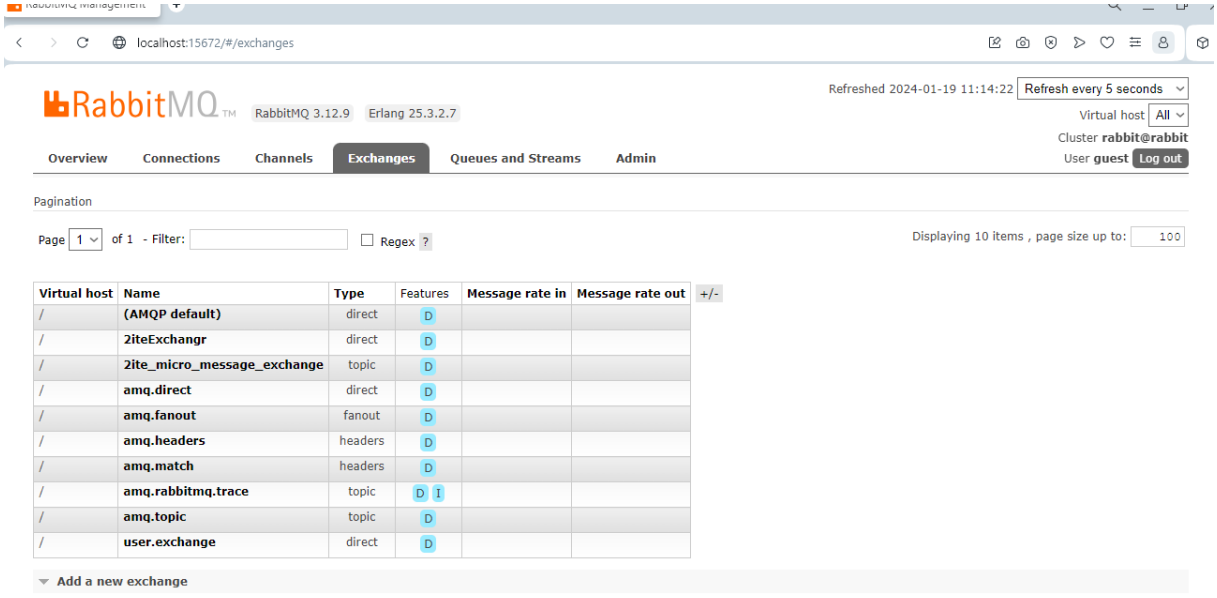
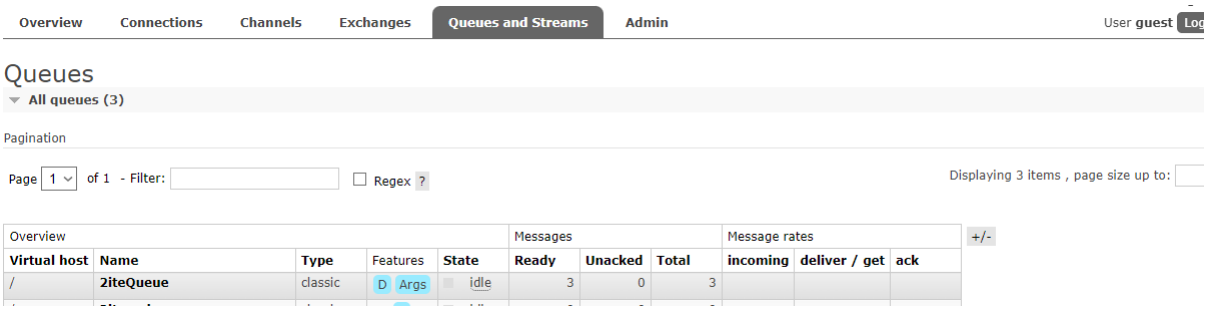
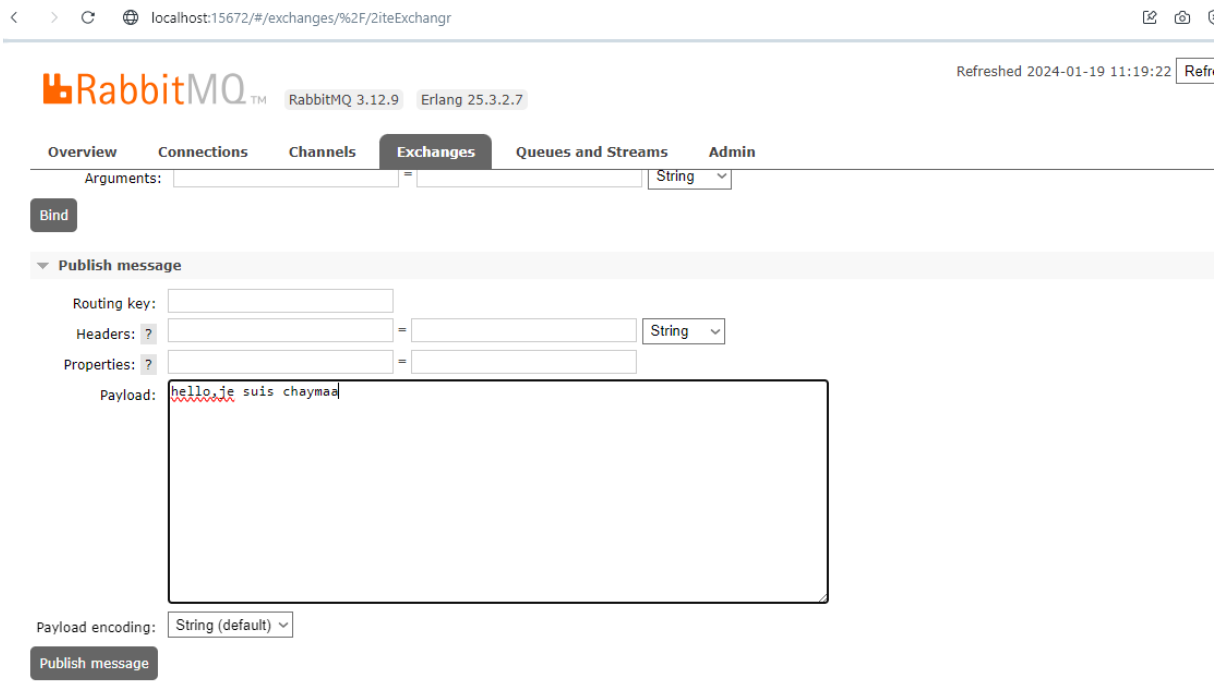
Interface :



Ajouter un Exchange : 2iteExchange



Ajouter un Queue : 2iteQueue



Publier un message

MicroservicesspringbotMessagerie

Dans cette section, nous allons examiner comment intégrer RabbitMQ dans Spring Boot en employant deux services distincts :

- **spring-rabbitmq-producer**
- **spring-rabbitmq-consumer** Au lieu de définir les paramètres de RabbitMQ dans l'interface comme nous l'avons fait précédemment, nous opterons cette fois-ci pour une configuration dynamique directement dans notre code. Cela nous permettra de tester la communication via Postman et de confirmer le bon fonctionnement de l'ensemble.

Structure du projet

application.properties

```
server.port = 8123
spring.rabbitmq.addresses = localhost:5672
```

Configuration RabbitMQ

```
import org.springframework.amqp.core.*;
import org.springframework.amqp.rabbit.connection.ConnectionFactory;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import
org.springframework.amqp.support.converter.Jackson2JsonMessageConverte
r;
import org.springframework.amqp.support.converter.MessageConverter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class MQConfig {
    public static final String QUEUE = "2ite_micro_message_queue";
    public static final String EXCHANGE =
"2ite_micro_message_exchange";
    public static final String ROUTING_KEY = "message_routingKey";

    @Bean
    public Queue queue() {
        return  new Queue(QUEUE);
    }

    @Bean
    public TopicExchange exchange() {
        return new TopicExchange(EXCHANGE);
    }

    @Bean
```

```
    public Binding binding(Queue queue, TopicExchange exchange) {
        return BindingBuilder
                .bind(queue)
                .to(exchange)
                .with(ROUTING_KEY);
    }

    @Bean
    public MessageConverter messageConverter() {
        return  new Jackson2JsonMessageConverter();
    }

    @Bean
    public AmqpTemplate template(ConnectionFactory connectionFactory)
{
        RabbitTemplate template = new
RabbitTemplate(connectionFactory);
        template.setMessageConverter(messageConverter());
        return  template;
    }


}
```

CustomMessage

```
package com.oussama.rabbitmicro;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

import java.util.Date;

@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class CustomMessage {

    private String messageId;
    private String message;
    private Date messageDate;

}
```

MessagePublisher

```java
package com.oussama.rabbitmicro;

import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import java.util.Date;
import java.util.UUID;

@RestController
public class MessagePublisher {

    @Autowired
    private RabbitTemplate template;

    @PostMapping("/publish")
    public String publishMessage(@RequestBody CustomMessage
message) {
        message.setMessageId(UUID.randomUUID().toString());
        message.setMessageDate(new Date());
        template.convertAndSend(MQConfig.EXCHANGE,
                MQConfig.ROUTING_KEY, message);

        return "Message Published";
    }
}
```

Ce code configure une application Spring Boot pour utiliser RabbitMQ en suivant les principes de la messagerie asynchrone.

1. **application.properties:** Fichier de configuration avec des propriétés, où **server.port** spécifie le port du serveur et **spring.rabbitmq.addresses** définit l'adresse de RabbitMQ.

2. **MQConfig (Configuration RabbitMQ):**

    - Définit les constantes telles que le nom de la file (**QUEUE**), le nom de l'échange (**EXCHANGE**), et la clé de routage (**ROUTING_KEY**).

    - Configure une file, un échange de type "topic", et la liaison entre eux.

    - Définit un convertisseur de message pour convertir les objets Java en JSON lors de la communication avec RabbitMQ.

    - Configure un template RabbitMQ pour envoyer des messages.

3. **CustomMessage:** Une classe Java simple pour représenter les messages personnalisés avec un identifiant, le contenu du message, et la date du message.

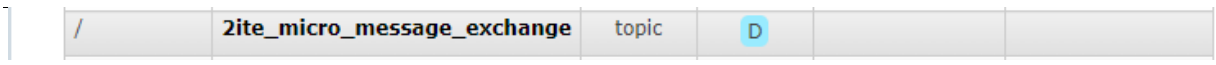
4. **MessagePublisher:**

    - Un contrôleur REST qui utilise le template RabbitMQ pour publier un message.

    - Lorsqu'une requête POST est reçue sur "/publish", il crée un objet **CustomMessage**, lui attribue un identifiant unique et la date actuelle, puis envoie ce message à l'échange RabbitMQ avec la clé de routage spécifiée.

En résumé, ce code met en place la configuration de RabbitMQ, définit un modèle de message personnalisé, et fournit une API REST (**MessagePublisher**) pour publier des messages dans RabbitMQ. Les messages publiés suivent la structure définie dans la classe **CustomMessage**.

Vérification dans l'interface



- ## spring-rabbitmq-cosomer

  - application.properties

```
server.port = 8223
spring.rabbitmq.addresses = localhost:5672
```

  - Configuration RabbitMQ

```
import org.springframework.amqp.core.*;
import
org.springframework.amqp.rabbit.connection.ConnectionFactory;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import
org.springframework.amqp.support.converter.Jackson2JsonMessageConve
rter;
import org.springframework.amqp.support.converter.MessageConverter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class MQConfig {
    public static final String QUEUE = "2ite_micro_message_queue";
    public static final String EXCHANGE =
"2ite_micro_message_exchange";
    public static final String ROUTING_KEY = "message_routingKey";

    @Bean
    public Queue queue() {
        return  new Queue(QUEUE);
    }

    @Bean
    public TopicExchange exchange() {
        return new TopicExchange(EXCHANGE);
    }
```

```
    @Bean
    public Binding binding(Queue queue, TopicExchange exchange) {
        return BindingBuilder
                .bind(queue)
                .to(exchange)
                .with(ROUTING_KEY);
    }

    @Bean
    public MessageConverter messageConverter() {
        return  new Jackson2JsonMessageConverter();
    }

    @Bean
    public AmqpTemplate template(ConnectionFactory
connectionFactory) {
        RabbitTemplate template = new
RabbitTemplate(connectionFactory);
        template.setMessageConverter(messageConverter());
        return  template;
    }
}
```

- CustomMessage

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

import java.util.Date;

@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class CustomMessage {

    private String messageId;
    private String message;
    private Date messageDate;

}
```

- MessageListner

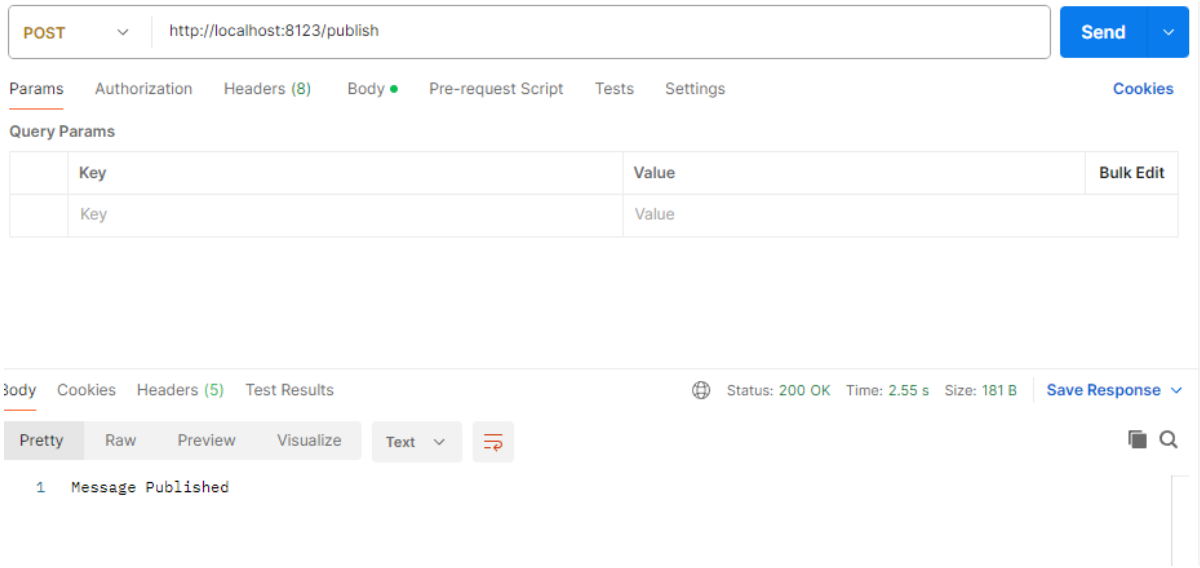
```java
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
public class MessageListener {

    @RabbitListener(queues = MQConfig.QUEUE)
    public void listener(CustomMessage message) {
        System.out.println(message);
    }

}
```

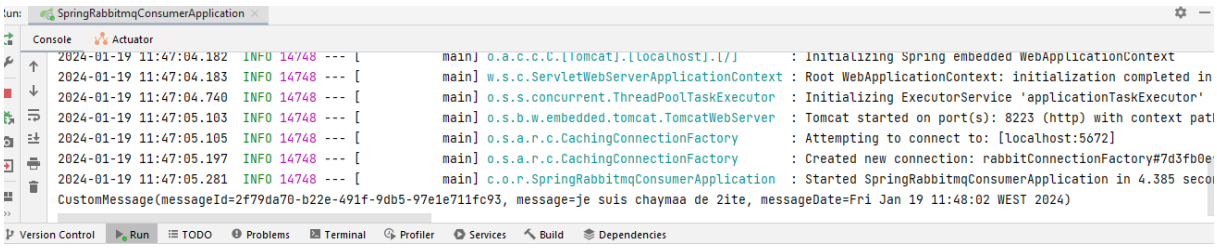
Ce code complète l'application RabbitMQ Spring Boot en ajoutant un composant **MessageListener** qui réagit aux messages publiés dans la file RabbitMQ. Voici une explication du nouveau code ajouté :

1. **application.properties**:
    - Le port du serveur est changé à 8223.
    - L'adresse de RabbitMQ reste localhost:5672.
2. **MessageListener**:
    - Un composant Spring (**@Component**) qui écoute les messages de la file définie dans **MQConfig** à l'aide de l'annotation **@RabbitListener**.
    - La méthode **listener** est appelée chaque fois qu'un message est placé dans la file. Elle prend un objet **CustomMessage** en paramètre, qui est automatiquement désérialisé à partir du format JSON du message.
    - Dans cet exemple, la méthode se contente d'afficher le message dans la console. Dans une application réelle, vous auriez probablement un traitement plus significatif ici.

En résumé, avec l'ajout de **MessageListener**, l'application est désormais capable de réagir aux messages publiés dans RabbitMQ et de prendre des mesures en conséquence. Cela crée une communication asynchrone entre le producteur (**MessagePublisher**) et le consommateur (**MessageListener**) via RabbitMQ.

TEST





Micro services springboot-MySQL

Dans cette section, nous mettrons en place le transfert de donnees du service producteur vers la base de donnees du service consommateur en exploitant la file d'attente de RabbitMQ. Cette approche permettra une communication asynchrone entre les services, ou les donnees seront envoyees a travers la file d'attente et traitees ulterieurement par le service consommateur pour leur insertion dans la base de donnees.

- ## spring-rabbitmq-producer

Configuration RabbitMQ

```
import
org.springframework.amqp.rabbit.connection.CachingConnectionFactory
;
import
org.springframework.amqp.rabbit.connection.ConnectionFactory;
import
org.springframework.amqp.support.converter.Jackson2JsonMessageConve
rter;
import org.springframework.amqp.support.converter.MessageConverter;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RabbitMQConfig {

    @Value("${spring.rabbitmq.host}")
    String host;

    @Value("${spring.rabbitmq.username}")
    String username;

    @Value("${spring.rabbitmq.password}")
    String password;

    @Bean
    CachingConnectionFactory connectionFactory() {
        CachingConnectionFactory cachingConnectionFactory = new
CachingConnectionFactory(host);
        cachingConnectionFactory.setUsername(username);
        cachingConnectionFactory.setPassword(password);
        return cachingConnectionFactory;
    }
```

```
    @Bean
    public MessageConverter jsonMessageConverter() {
        return new Jackson2JsonMessageConverter();
    }

    @Bean
    public RabbitTemplate rabbitTemplate(ConnectionFactory
connectionFactory) {
        final RabbitTemplate rabbitTemplate = new
RabbitTemplate(connectionFactory);
        rabbitTemplate.setMessageConverter(jsonMessageConverter());
        return rabbitTemplate;
    }
}
```

Classe User

```
import com.fasterxml.jackson.annotation.JsonIdentityInfo;
import com.fasterxml.jackson.annotation.ObjectIdGenerators;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;
import org.springframework.stereotype.Component;

import java.io.Serializable;

@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
@Component
public class User implements Serializable {
    private String userId;
    private String userName;
}
```

ProducerController

```
import microservices.messaging.domain.User;
import microservices.messaging.service.ProducerService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/")
public class ProducerController {

    private ProducerService producerService;
    private static final Logger logger =
LoggerFactory.getLogger(ProducerController.class);

    @Autowired
    public ProducerController(ProducerService producerService) {
        this.producerService = producerService;
    }

    @Value("${app.message}")
    private String response;

    @PostMapping("/produce")
    public ResponseEntity<String> sendMessage(@RequestBody User
user) {
        producerService.sendMessage(user);
        logger.info("user sent: " + user);
        return ResponseEntity.ok("user sent: " + user);
    }
}
```

ProducerService

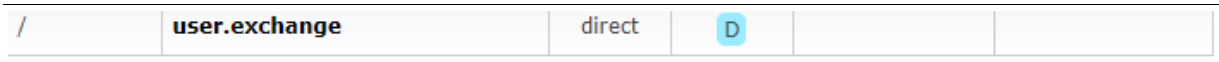
```
package microservices.messaging.service;

import microservices.messaging.domain.User;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;

@Service
public class ProducerService {

    private RabbitTemplate rabbitTemplate;

    @Autowired
    public ProducerService(RabbitTemplate rabbitTemplate) {
        this.rabbitTemplate = rabbitTemplate;
    }

    @Value("${spring.rabbitmq.exchange}")
    private String exchange;

    @Value("${spring.rabbitmq.routingkey}")
    private String routingkey;

    public void sendMessage(User user) {
        rabbitTemplate.convertAndSend(exchange,routingkey, user);
    }

}
```

Application.yml

```yaml
app:
  message: message succesfuly sent
spring:
  rabbitmq:
    host: localhost
    password: guest
    port: 15672
    username: guest
    exchange: user.exchange
    routingkey: user.routingkey
server:
  port: 8081
```

ce code configure la communication entre microservices en utilisant RabbitMQ. Le service producteur (**ProducerService**) envoie un utilisateur à RabbitMQ, et le contrôleur (**ProducerController**) expose un point de terminaison pour déclencher ce processus. La configuration RabbitMQ est centralisée dans la classe **RabbitMQConfig**.

Vérification dans l'interface



- ## spring-rabbitmq-cosomer

  - application.properties

```
spring:
  rabbitmq:
    host: localhost
    password: guest
    port: 15672
    username: guest
    exchange: user.exchange
    queue: user.queue
    routingkey: user.routingkey
```

  - application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/testdb?useSSL=fal
se&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
```

  - Configuration RabbitMQ

```
package microservices.messagingconsumer.config;
```

```
import org.springframework.amqp.core.*;
import
org.springframework.amqp.rabbit.connection.CachingConnectionFactory
;
import
org.springframework.amqp.rabbit.connection.ConnectionFactory;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import
org.springframework.amqp.support.converter.Jackson2JsonMessageConve
rter;
import org.springframework.amqp.support.converter.MessageConverter;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RabbitMQConfig {

    @Value("${spring.rabbitmq.queue}")
    private String queue;

    @Value("${spring.rabbitmq.exchange}")
    private String exchange;

    @Value("${spring.rabbitmq.routingkey}")
    private String routingKey;

    @Value("${spring.rabbitmq.username}")
    private String username;

    @Value("${spring.rabbitmq.password}")
    private String password;

    @Value("${spring.rabbitmq.host}")
    private String host;

    @Bean
    Queue queue() {
        return new Queue(queue, true);
    }

    @Bean
    Exchange myExchange() {
        return
ExchangeBuilder.directExchange(exchange).durable(true).build();
    }

    @Bean
    Binding binding() {
        return BindingBuilder
                .bind(queue())
                .to(myExchange())
                .with(routingKey)
                .noargs();
    }

    @Bean
    public ConnectionFactory connectionFactory() {
        CachingConnectionFactory cachingConnectionFactory = new
CachingConnectionFactory(host);
        cachingConnectionFactory.setUsername(username);
```

```
            cachingConnectionFactory.setPassword(password);
            return cachingConnectionFactory;
        }

        @Bean
        public MessageConverter jsonMessageConverter() {
            return new Jackson2JsonMessageConverter();
        }

        @Bean
        public RabbitTemplate rabbitTemplate(ConnectionFactory
    connectionFactory) {
            final RabbitTemplate rabbitTemplate = new
    RabbitTemplate(connectionFactory);
            rabbitTemplate.setMessageConverter(jsonMessageConverter());
            return rabbitTemplate;
        }
    }


    }
```

- CustomMessage

```
    import lombok.AllArgsConstructor;
    import lombok.Data;
    import lombok.NoArgsConstructor;
    import lombok.ToString;

    import java.util.Date;

    @Data
    @NoArgsConstructor
    @AllArgsConstructor
    @ToString
    public class CustomMessage {

        private String messageId;
        private String message;
        private Date messageDate;

    }
```

- User Class

```
package microservices.messagingconsumer.domain;

import com.fasterxml.jackson.annotation.JsonIdentityInfo;
import com.fasterxml.jackson.annotation.ObjectIdGenerators;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;
import org.springframework.stereotype.Component;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import java.io.Serializable;

@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
@Entity
public class User implements Serializable {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String userId;
    private String userName;
}
```

- User Repository

```java
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
}
```

- Consumer Service

```java
import org.springframework.stereotype.Service;

@Service
public class ConsumerService {

    private final UserRepository userRepository;
    private static final Logger logger =
LoggerFactory.getLogger(ConsumerService.class);

    @Autowired
    public ConsumerService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @RabbitListener(queues = "${spring.rabbitmq.queue}")
    public void receivedMessage(User user) {
        User save = userRepository.save(user);
        logger.info("persisted " + save);
        logger.info("User recieved: " + user);
    }

}
```

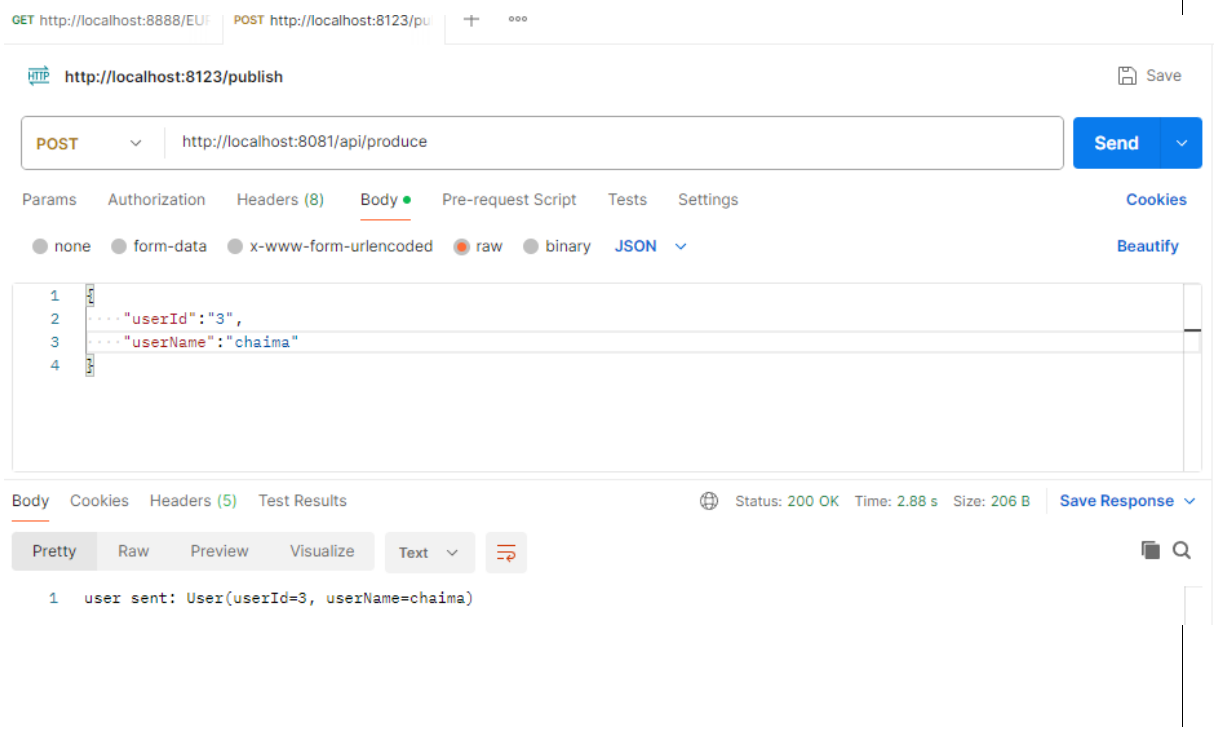
1. **RabbitMQConfig** (Configuration RabbitMQ) :
    - Définit la configuration RabbitMQ à l'aide des propriétés lues depuis le fichier **application.properties**.
    - Crée une file, un échange direct, et une liaison entre eux.
    - Configure une fabrique de connexion RabbitMQ avec les paramètres

spécifiés.

- Définit un convertisseur de message JSON.
- Configure un template RabbitMQ pour l'envoi et la réception de messages.

2. **CustomMessage** :
    - Représente un message personnalisé avec un identifiant, un contenu de message, et une date.
3. **User** :
    - Entité JPA représentant un utilisateur avec un identifiant généré automatiquement, un identifiant d'utilisateur et un nom d'utilisateur.
4. **UserRepository** :
    - Interface JPA pour la gestion des opérations de base de données pour l'entité **User**.
5. **ConsumerService** :
    - Service annoté avec **@RabbitListener** pour écouter les messages de la file RabbitMQ spécifiée.
    - Persiste l'utilisateur reçu dans la base de données en utilisant le **UserRepository**.
    - Utilise le logger SLF4J pour enregistrer les informations sur les opérations effectuées.

Test :


GET http://localhost:8888/EUI
POST http://localhost:8123/pu
http://localhost:8123/publish
Save
POST
http://localhost:8081/api/produce
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
JSON
Beautify
{
"userId":"3",
"userName":"chaima"
}
Body
Cookies
Headers (5)
Test Results
Status: 200 OK
Time: 2.88 s
Size: 206 B
Save Response
Pretty
Raw
Preview
Visualize
Text
user sent: User(userId=3, userName=chaima)


Run:
MessagingApplication
MessagingConsumerApplication
Console
Actuator
Running...
2024-01-19 12:17:28.760 INFO 13756 --- [nio-8081-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet
2024-01-19 12:17:28.775 INFO 13756 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-01-19 12:17:28.865 INFO 13756 --- [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 90 ms
2024-01-19 12:17:29.726 INFO 13756 --- [nio-8081-exec-1] o.s.a.r.c.CachingConnectionFactory : Attempting to connect to: localhost:5672
2024-01-19 12:17:30.157 INFO 13756 --- [nio-8081-exec-1] o.s.a.r.c.CachingConnectionFactory : Created new connection: connectionFactory#2eae4349:0/Sim
2024-01-19 12:17:30.231 INFO 13756 --- [nio-8081-exec-1] o.m.m.controller.ProducerController : user sent: User(userId=3, userName=chaima)