Microservice with RabbitMQ

Learning outcome

you will implement spring-boot microservice to achieve the following

- Produce and consume messages using rabbitmq

Scenario

Order microservice pruduces some message and inventory service receives those message.

Solution

Below some outline is given to implement the application.

- Install Docker https://docs.docker.com/get-docker/
- Execute following command to run rabbitmq in docker

```
docker run -it --hostname my-rabbit -p 15672:15672 -p 5672:5672 rabbitmq:3-management
```

Go to the browser. Hit localhost:15672 to see RabbitMQ dashboard Use guest as username and guest as password

Then, build 2 services one will produce some messages and another will consume those messages

Suppose, OrderService will produce message of order. Add the following code to create order service

OrderService

create following packages - entity - config - controller

In entity package, create order class

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class Order {

    private String orderId;
    private String name;
    private int qty;
    private double price;

}
```

Also, in entity create order status class to send order and some additional information to rabbitmq

```
@AllArgsConstructor
@Data
@NoArgsConstructor
@ToString
public class OrderStatus {

    private Order order;
    private String status;
    private String message;
}
```

In config package, add MessagingConfig class

```
@Configuration
public class MessagingConfig {

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

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

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

You will notice there are some constant values required to configure rabbitmq
Create a class Constants in the root package

```
public class Constants {
    public static final String QUEUE = "rabbit_queue";
    public static final String EXCHANGE = "rabbit_exchange";
    public static final String ROUTING_KEY = "rabbit_routingKey";
}
```

Now, in the controller package create OrderController that will produce massage to rabbitmq. Here, request will come from postman with order information

```
@RestController
@RequestMapping("/order")
public class OrderController {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    @PostMapping("/{restaurantName}")
    public String bookOrder(@RequestBody Order order, @PathVariable String restaurantName )
        order.setOrderId(UUID.randomUUID().toString());
        OrderStatus orderStatus = new OrderStatus(order, "PROCESS", "Order Successfully Plac

        rabbitTemplate.convertAndSend(Constants.EXCHANGE,Constants.ROUTING_KEY, orderStatus)
        return "success!!";
    }
}
```

In the application.properties configure for server port and rabbitmq

```
server.port = 8001
spring.rabbitmq.addresses = localhost:5672
```

Send some request from postman json

```
{
"name" : "barger",
"qty" : 2,
"price" : 500
}
```

request to

```
localhost:8001/order/bismillah
```

Now, if you check in the browser localhost:15672 you will see some messages in the queue

Inventory service

To create another service say Inventory service. Do the same thing as above services except controller.

Create an Inventory controller class which will consume the message. As a controller it can perform different things. now use this as a consumer of the message. Give a different port for this service.

```
@Component
public class InventoryController {

    @RabbitListener(queues = Constants.QUEUE )
    public void consumeMessageFromQueue(OrderStatus orderStatus) {
        System.out.println("Message Received from queue: " +orderStatus );
    }
}
```

Node code

if you want to implement it in node and express. you can add following code for rabbitmq to produce and consume the messages along with other code.

you can install amqplib using the following command

```
npm install amqplib
```

For producer

```
const rabbit = require('amqplib');
const QUEUE_NAME = 'square';
const EXCHANGE_TYPE = 'direct';
const EXCHANGE_NAME = 'main';
const KEY = 'myKey';
const message = 'something'
connection = rabbit.connect('amqp://localhost');
connection.then(async (conn)=>{
   const channel = await conn.createChannel();
   await channel.assertExchange(EXCHANGE_NAME, EXCHANGE_TYPE);
   await channel.assertQueue(QUEUE_NAME);
   channel.bindQueue(QUEUE_NAME, EXCHANGE_NAME, KEY);
   channel.sendToQueue(QUEUE_NAME, Buffer.from(message))
})
```

For consumer

```
const rabbit = require('amqplib');
const QUEUE_NAME = 'square';
connection = rabbit.connect('amqp://localhost');
connection.then(async (conn)=>{
   const channel = await conn.createChannel();
   channel.consume(QUEUE_NAME, (m)=>{
       const number = parseInt(m.content.toString())
       const square = number * number
```

```
        console.log(square)
        channel.ack(m)
    })
})
```

Task

- Create a different message from the order service, say a product id will be added with the orderStatus.
- Create a Product entity in the inventory service with (id, name, quantity);
- Check whether order can be fullfilled by checking the product quantity in the inventory service or not and print a message.

THANK YOU!