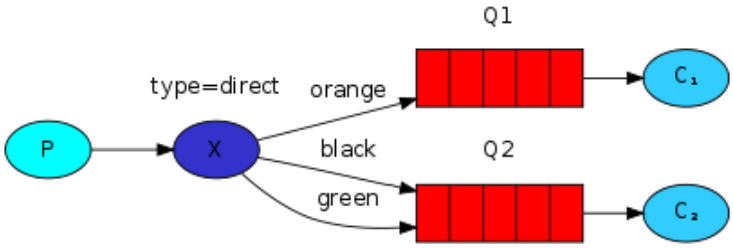
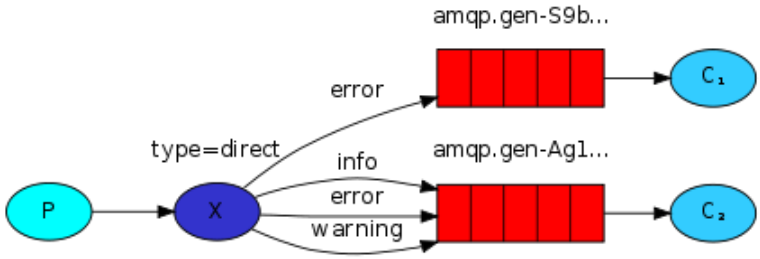
Routing

Direct exchange

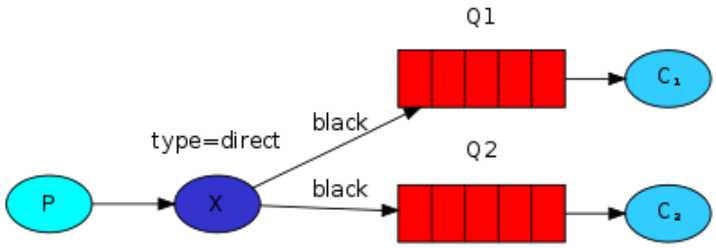
Quando utilizamos uma exchange do tipo **fanout** todas as mensagens são enviadas para todos os consumidores, exchange do tipo **fanout** não possui muita flexibilidade pois só permite **broadcasting** de mensagens a todos os consumidores.

Para fazer o roteamento pode-se utilizar o tipo **direct** para configuração da exchange, onde a mensagem será enviada para a queue que possuam a **binding-key** que realize o match com a **routing-key** da mensagem.





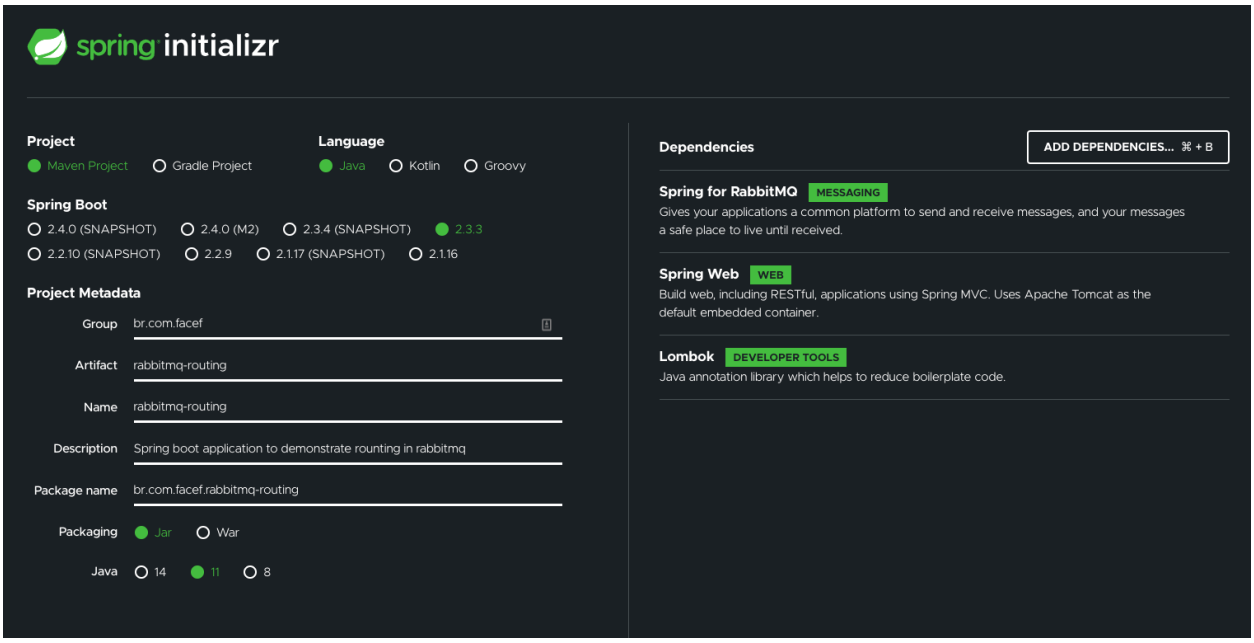
Multiple bindings



É possível realizar o **bind** para múltiplas queues com a mesma **binding-key**. No exemplo da imagem acima, as filas **Q1 e Q2** estão configuradas para fazer o binding com a exchange **X** do tipo **direct** através da binding-key **black**, nesse exemplo a exchange irá se comportar como uma exchange do tipo **fanout**, pois irá enviar as mensagens para as filas **Q1 e Q2.**

Exemplo

Para gerar o projeto podem acessar a **URL**, nessa url iremos utilizar o **Spring Initializr** para gerar a estrutura padrão do projeto já adicionando a biblioteca do RabbitMQ.



Primeiramente precisamos iniciar o RabbitMQ:

- Iniciando diretamente via docker

```
docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3-management
```

- Iniciando via docker-compose

```
docker-compose stop && docker-compose rm -f && docker-compose up -d
```

- Iniciando via helm + microk8s

```
microk8s helm3 install rabbitmq stable/rabbitmq
```

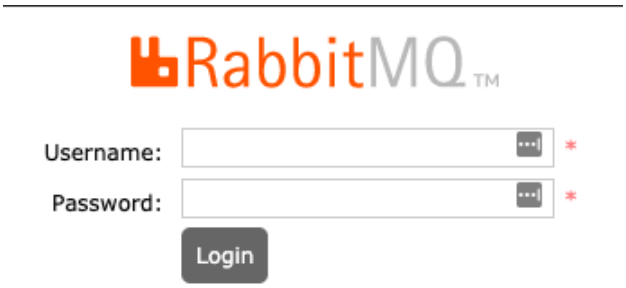
Para acompanhar a inicialização do container pode-se utilizar o comando:

```
docker-compose logs -f
```

Verificando os logs:

```
rabbitmq_1  | 2020-08-25 22:59:22.925 [info] <0.675.0> Ready to start client connection listeners
rabbitmq_1  | 2020-08-25 22:59:22.930 [info] <0.980.0> started TCP listener on [::]:5672
rabbitmq_1  | 2020-08-25 22:59:23.273 [info] <0.675.0> Server startup complete; 4 plugins started.
rabbitmq_1  |  * rabbitmq_prometheus
rabbitmq_1  |  * rabbitmq_management
rabbitmq_1  |  * rabbitmq_web_dispatch
rabbitmq_1  |  * rabbitmq_management_agent
rabbitmq_1  |  completed with 4 plugins.
```

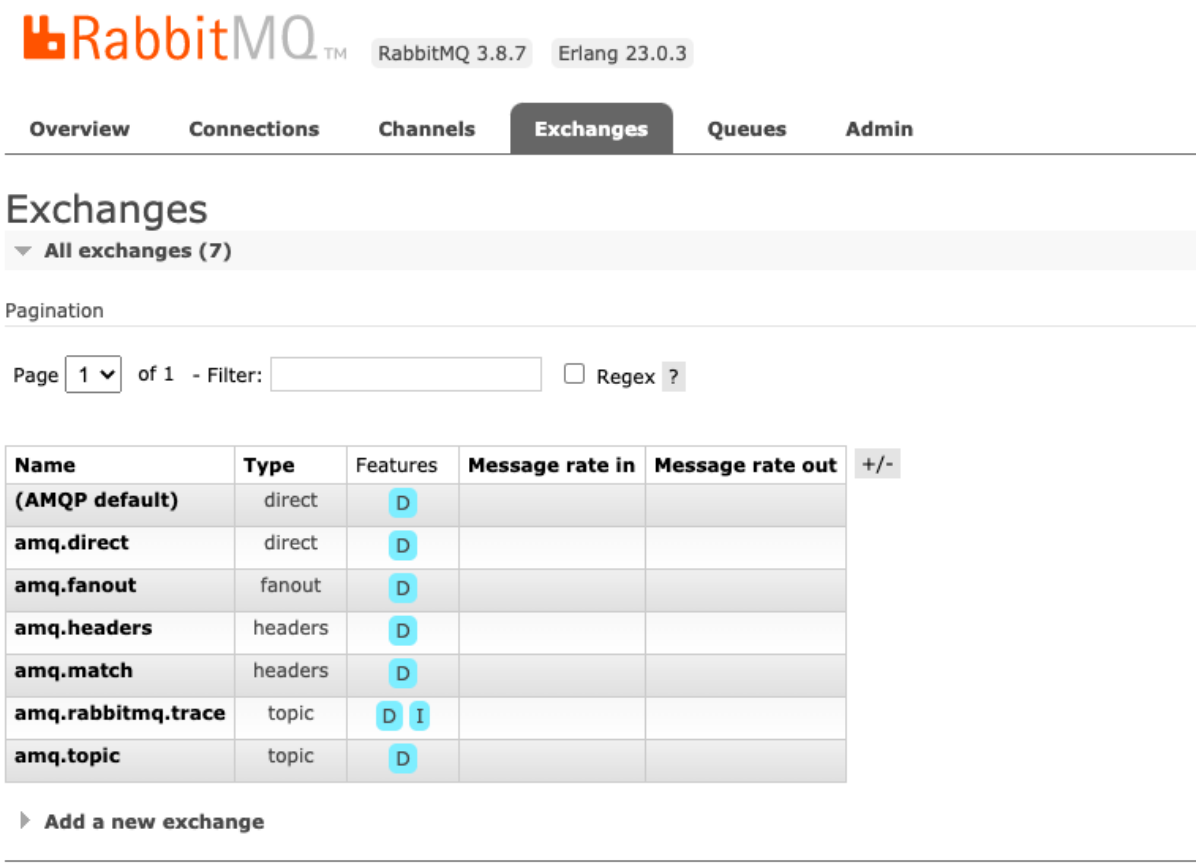
Após o início do container, podemos acessar a URL do admin através do endereço http://localhost:15672/
username: guest
password: guest



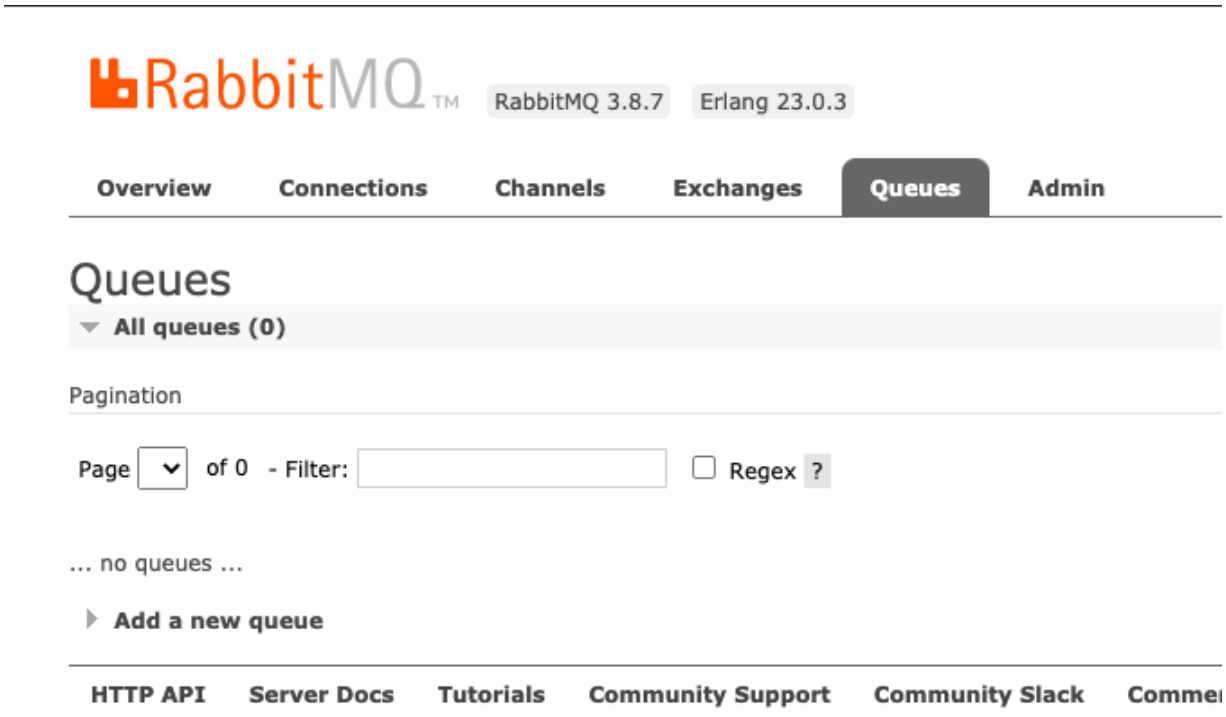
Após o login temos acesso a interface de gerenciamento do servidor do rabbitmq, onde temos as visões referentes há:

- **Overview**
- **Connections**
- **Channels**
- **Exchanges**
- **Queues**
- **Admin**

Ao clicar em **exchanges** podemos visualisar as exchanges padrões que o próprio servidor o rabbitmq cria quando iniciamos o serviço. E temos a opção de criar nossas próprias exchanges através do admin.



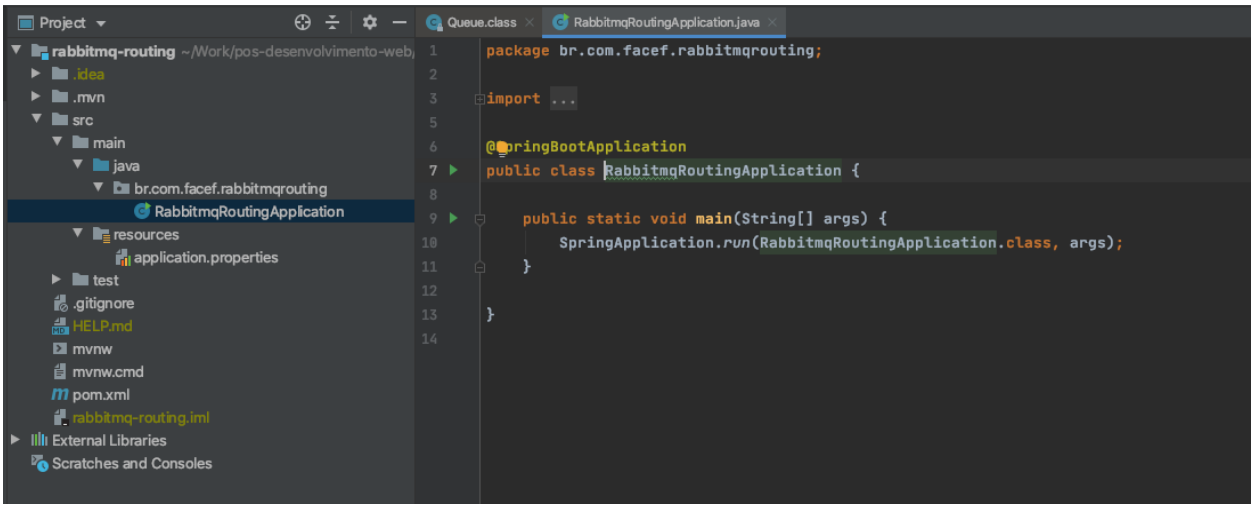
Clicando em **queues** podemos visualizar que por default nenhuma fila é criada no momento da inicialização do servidor.



Para demonstrar o funcionamento, irei fazer um passo a passo com um exemplo funcional, implementando uma API Rest que recebe uma mensagem via POST com um body que será enviado para a exchange criada, e com base no dado enviado no body a mensagem será roteada para a queue em específico.

Passo a passo:

- **Initial project**



- **Include docker-compose**

```
version: '3'
services:
  rabbitmq:
    image: rabbitmq:3-management
    ports:
      - "5672:5672"
      - "15672:15672"
```

- **Include rabbitmq configuration to DirectExchange**

```
package br.com.facef.rabbitmqrouting.configuration;

import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.BindingBuilder;
import org.springframework.amqp.core.DirectExchange;
import org.springframework.amqp.core.ExchangeBuilder;
import org.springframework.amqp.core.Queue;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class DirectExchangeConfiguration {

  public static final String DIRECT_EXCHANGE_NAME = "order-exchange";
  public static final String PAYMENT_CREDITCARD_QUEUE_NAME = "payment-creditcard-queue";
  public static final String PAYMENT_BANKSLIP_QUEUE_NAME = "payment-bankslip-queue";

  @Bean
  Queue paymentCreditCardQueue() {
    return new Queue(PAYMENT_CREDITCARD_QUEUE_NAME);
  }

  @Bean
  Queue paymentBankSlipQueue() {
    return new Queue(PAYMENT_BANKSLIP_QUEUE_NAME);
  }

  @Bean
  DirectExchange exchange() {
    return ExchangeBuilder.directExchange(DIRECT_EXCHANGE_NAME).durable(true).build();
  }

  @Bean
  Binding bindingPaymentCreditCardQueue(
      @Qualifier("paymentCreditCardQueue") Queue queue, DirectExchange exchange) {
    return BindingBuilder.bind(queue).to(exchange).with("creditcard");
  }

  @Bean
  Binding bindingPaymentBankSlipQueue(
      @Qualifier("paymentBankSlipQueue") Queue queue, DirectExchange exchange) {
    return BindingBuilder.bind(queue).to(exchange).with("bankslip");
```

```
  }
}
```

- **Include DTO Message class to store data**

```
package br.com.facef.rabbitmqrouting.dto;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.ToString;

@AllArgsConstructor
@Getter
@ToString
public class Message {

  private String orderId;
  private String paymentType;
}
```

- **Include service class to send message to rabbitmq**

```
package br.com.facef.rabbitmqrouting.service;

import br.com.facef.rabbitmqrouting.configuration.DirectExchangeConfiguration;
import br.com.facef.rabbitmqrouting.dto.Message;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class MessageService {

  @Autowired private RabbitTemplate rabbitTemplate;

  public void sendToDirectExchange(Message message) {
    try {
      final var messageJson = new ObjectMapper().writeValueAsString(message);

      rabbitTemplate.convertAndSend(
          DirectExchangeConfiguration.DIRECT_EXCHANGE_NAME, getRoutingKey(message), messageJson);
    } catch (JsonProcessingException e) {
      throw new RuntimeException(e);
    }
  }

  private String getRoutingKey(Message message) {
    switch (message.getPaymentType()) {
      case "creditCard":
        return "creditcard";
      case "bankSlip":
        return "bankslip";
```

```
      default:
        throw new IllegalArgumentException("Invalid paymentType");
    }
  }
}
```

- **Create a controller to receive message and send to a rabbitmq**

```
package br.com.facef.rabbitmqrouting.controller;

import br.com.facef.rabbitmqrouting.dto.Message;
import br.com.facef.rabbitmqrouting.service.MessageService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/routing")
public class MessagesController {

  @Autowired private MessageService messageService;

  @PostMapping("/direct")
  public ResponseEntity placeOrder(@RequestBody Message message) {
    messageService.sendToDirectExchange(message);
    return ResponseEntity.accepted().build();
  }
}
```

Após a finalização da implementação do projeto iremos executar o projeto utilizando o próprio plugin do spring.

```
./mvnw clean spring-boot:run
```

E com isso iremos realizar os testes via **Postmam ou Curl** chamando a API para inclusão das mensagens nas filas e verificar o comportamento do roteamento.

Realizando uma requisição com paymentType **CreditCard**

```
curl --location --request POST 'http://localhost:8080/routing/direct' \
--header 'Content-Type: application/json' \
--data-raw '{
    "orderId": "839128391283",
    "paymentType": "creditCard"
}'
```

Realizando uma requisição com paymentType **BankSlip**

```
curl --location --request POST 'http://localhost:8080/routing/direct' \
--header 'Content-Type: application/json' \
--data-raw '{
    "orderId": "839128391283",
    "paymentType": "bankSlip"
}'
```

Após realizarmos a primeira chamada via API, podemos verificar que temos uma nova exchange criada chamada **order-exchange**.

RabbitMQ™ RabbitMQ 3.8.7 Erlang 23.0.3

Overview | Connections | Channels | Exchanges | Queues | Admin

Exchanges

All exchanges (8)

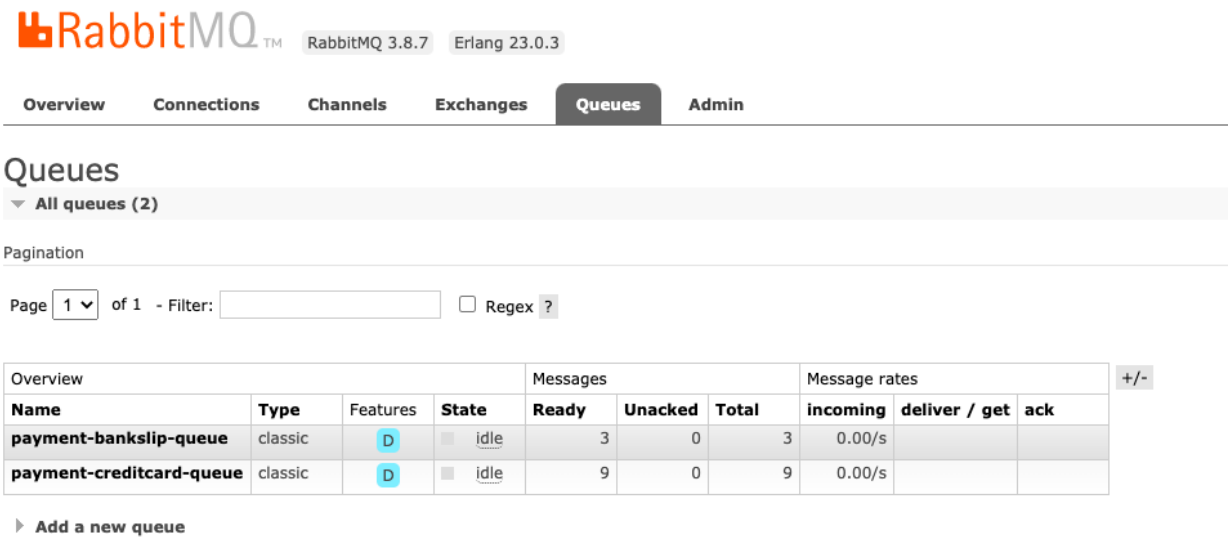
Pagination

Page 1 of 1 - Filter: Regex ?

Name	Type	Features	Message rate in	Message rate out	+/-
(AMQP default)	direct	D			
amq.direct	direct	D			
amq.fanout	fanout	D			
amq.headers	headers	D			
amq.match	headers	D			
amq.rabbitmq.trace	topic	D I			
amq.topic	topic	D			
order-exchange	direct	D	0.00/s	0.00/s	

Add a new exchange

E também temos a visão das filas que foram criadas para a exchange **order-exchange** já com as mensagens que enviamos via API.



Para demonstrar o funcionamento do consumo em cada **queue**, iremos criar uma classe reponsável por fazer o consumo das mensagens e imprimir no console o log com o seu conteúdo.

- **Create a consumer to processing messages**

```
package br.com.facef.rabbitmqrouting.consumer;

import br.com.facef.rabbitmqrouting.configuration.DirectExchangeConfiguration;
import lombok.extern.slf4j.Slf4j;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

@Component
@Slf4j
public class MessageConsumer {

  @RabbitListener(queues = DirectExchangeConfiguration.PAYMENT_CREDITCARD_QUEUE_NAME)
  public void consumeCreditCardQueue(Message message) {
    log.info("Message processed from CreditCard Queue {}", new String(message.getBody()));
  }

  @RabbitListener(queues = DirectExchangeConfiguration.PAYMENT_BANKSLIP_QUEUE_NAME)
  public void consumeBankSlipQueue(Message message) {
    log.info("Message processed from BankSlip Queue {}", new String(message.getBody()));
  }
}
```

Na saída do console conseguimos identificar o consumo das mensagens corretamente por fila:

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.3.RELEASE)

2020-08-27 00:00:28.139  INFO 12434 --- [            main] b.c.f.r.RabbitmqRoutingApplication       : Starting RabbitmqRoutingApplication on Cassios-MacBook-Air.local with PID 12434 (/Users/cassiothadeu/Work/pos-desenvolvimento-web/mensageria-stre
s/rabbitmq-routing/target/classes started by cassiothadeu in /Users/cassiothadeu/Work/pos-desenvolvimento-web/mensageria-streams/rabbitmq-routing)
2020-08-27 00:00:28.143  INFO 12434 --- [            main] b.c.f.r.RabbitmqRoutingApplication       : No active profile set, falling back to default profiles: default
2020-08-27 00:00:30.371  INFO 12434 --- [            main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2020-08-27 00:00:30.412  INFO 12434 --- [            main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2020-08-27 00:00:30.413  INFO 12434 --- [            main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.37]
2020-08-27 00:00:30.617  INFO 12434 --- [            main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2020-08-27 00:00:30.618  INFO 12434 --- [            main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2341 ms
2020-08-27 00:00:31.630  INFO 12434 --- [            main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2020-08-27 00:00:32.065  INFO 12434 --- [            main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2020-08-27 00:00:32.069  INFO 12434 --- [            main] o.s.a.r.c.CachingConnectionFactory       : Attempting to connect to: [localhost:5672]
2020-08-27 00:00:32.283  INFO 12434 --- [            main] o.s.a.r.c.CachingConnectionFactory       : Created new connection: rabbitConnectionFactory#40e60ece:0/SimpleConnection@7b32b129 [delegate=amqp://guest@127.0.0.1:5672/, localPort= 54843]
2020-08-27 00:00:32.445  INFO 12434 --- [            main] b.c.f.r.RabbitmqRoutingApplication       : Started RabbitmqRoutingApplication in 10.499 seconds (JVM running for 11.65)
2020-08-27 00:00:34.906  INFO 12434 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2020-08-27 00:00:34.906  INFO 12434 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2020-08-27 00:00:34.921  INFO 12434 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 15 ms
2020-08-27 00:00:35.137  INFO 12434 --- [ntContainer#0-1] b.c.f.r.consumer.MessageConsumer         : Message processed from CreditCard Queue {"orderId":"839128391283","paymentType":"creditCard"}
2020-08-27 00:00:45.710  INFO 12434 --- [ntContainer#1-1] b.c.f.r.consumer.MessageConsumer         : Message processed from BankSlip Queue {"orderId":"839128391283","paymentType":"bankSlip"}
```