SPRING BOOT + ANGULAR +STRIPE

# PAGOS CON STRIPE EN SPRING BOOT Y ANGULAR 8. CAPÍTULO 4: STRIPE

Y por fin llegamos al meollo de la cuestión, donde vamos a comenzar con Stripe. Lo primero que vamos a hacer es implementar el servicio, como veremos a continuación es bastante sencillo. Pero antes de ir al servicio comenzaremos con la clase **PaymentIntentDto**:

|  |  |
| --- | --- |
|  | package com.cavanosa.TutorialStripe.http; |
|  |  |
|  | public class PaymentIntentDto { |
|  | public enum Currency{ |
|  | USD, EUR; |
|  | } |
|  |  |
|  | private String description; |
|  | private int amount; |
|  | private Currency currency; |
|  |  |
|  | public String getDescription() { |
|  | return description; |
|  | } |
|  |  |
|  | public void setDescription(String description) { |
|  | this.description = description; |
|  | } |
|  |  |
|  | public int getAmount() { |
|  | return amount; |
|  | } |
|  |  |
|  | public void setAmount(int amount) { |
|  | this.amount = amount; |
|  | } |
|  |  |
|  | public Currency getCurrency() { |
|  | return currency; |
|  | } |
|  |  |
|  | public void setCurrency(Currency currency) { |
|  | this.currency = currency; |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/cavanosa/a41e8124b3274f95935d520c095a482f/raw/b96a04a35cd56f52f744f1b514cbb29a85a2df4f/paymentintentdto.java)[**paymentintentdto.java**](https://gist.github.com/cavanosa/a41e8124b3274f95935d520c095a482f#file-paymentintentdto-java) hosted with  by [**GitHub**](https://github.com/)

Aquí tenemos un tipo enumerado -Currency- para la moneda que se vaya a utilizar. Los atributos son los que se van pasar como parámetro al servicio, que llegarán desde el cliente.

Y ahora, una vez que lo tenemos, vamos a **PaymentService**:

|  |  |
| --- | --- |
|  | package com.cavanosa.TutorialStripe.service; |
|  |  |
|  | import com.cavanosa.TutorialStripe.http.PaymentIntentDto; |
|  | import com.stripe.Stripe; |
|  | import com.stripe.exception.StripeException; |
|  | import com.stripe.model.PaymentIntent; |
|  | import org.springframework.beans.factory.annotation.Value; |
|  | import org.springframework.stereotype.Service; |
|  |  |
|  | import java.util.ArrayList; |
|  | import java.util.HashMap; |
|  | import java.util.Map; |
|  |  |
|  | @Service |
|  | public class PaymentService { |
|  |  |
|  | @Value("${stripe.key.secret}") |
|  | String secretKey; |
|  |  |
|  | public PaymentIntent paymentIntent(PaymentIntentDto paymentIntentDto) throws StripeException { |
|  | Stripe.apiKey = secretKey; |
|  | Map<String, Object> params = new HashMap<>(); |
|  | params.put("amount", paymentIntentDto.getAmount()); |
|  | params.put("currency", paymentIntentDto.getCurrency()); |
|  | params.put("description", paymentIntentDto.getDescription()); |
|  | ArrayList payment\_method\_types = new ArrayList(); |
|  | payment\_method\_types.add("card"); |
|  | params.put("payment\_method\_types", payment\_method\_types); |
|  | return PaymentIntent.create(params); |
|  | } |
|  |  |
|  | public PaymentIntent confirm(String id) throws StripeException { |
|  | Stripe.apiKey = secretKey; |
|  | PaymentIntent paymentIntent = PaymentIntent.retrieve(id); |
|  | Map<String, Object> params = new HashMap<>(); |
|  | params.put("payment\_method", "pm\_card\_visa"); |
|  | paymentIntent.confirm(params); |
|  | return paymentIntent; |
|  | } |
|  |  |
|  | public PaymentIntent cancel(String id) throws StripeException { |
|  | Stripe.apiKey = secretKey; |
|  | PaymentIntent paymentIntent = PaymentIntent.retrieve(id); |
|  | paymentIntent.cancel(); |
|  | return paymentIntent; |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/cavanosa/85b68e73a24df4f03d920205364805ee/raw/d82e180b1dbde2b8c515c6644814292c1a5d2e8c/paymentservice.java)[**paymentservice.java**](https://gist.github.com/cavanosa/85b68e73a24df4f03d920205364805ee#file-paymentservice-java) hosted with  by [**GitHub**](https://github.com/)

Accedemos a nuestra clave secreta con la anotación @Value (la tenemos en application.properties).

Para realizar un pago utilizamos la clave PaymentIntent, a la que le pasaremos varios parámetros con un Map . Entre ellos están la cantidad a pagar (amount), el tipo de moneda (currency) o el método de pago. Stripe admite varios, por eso este parámetro se lo pasamos con un ArrayList. En este ejemplo vamos a utilizar la tarjeta (card).

Si vamos a la [documentación](https://stripe.com/docs/api/payment_intents/object)podemos ver todos los atributos del objeto paymentintent.

Tenemos dos métodos más, uno para confirmar el pago y otro para cancelarlo. En ambos se obtiene el PaymentIntent a partir de su id con el método retrieve. En el caso de confirmar hay que añadir el id del método de pago, tarjeta en nuestro caso (params.put(“payment\_method”, “pm\_card\_visa”))

**Controlador:**

Y ya para finalizar el backend implementamos StripeController:

|  |  |
| --- | --- |
|  | package com.cavanosa.TutorialStripe.controller; |
|  |  |
|  | import com.cavanosa.TutorialStripe.http.PaymentIntentDto; |
|  | import com.cavanosa.TutorialStripe.service.PaymentService; |
|  | import com.stripe.exception.StripeException; |
|  | import com.stripe.model.PaymentIntent; |
|  | import org.springframework.beans.factory.annotation.Autowired; |
|  | import org.springframework.http.HttpStatus; |
|  | import org.springframework.http.ResponseEntity; |
|  | import org.springframework.web.bind.annotation.\*; |
|  |  |
|  | @RestController |
|  | @RequestMapping("/stripe") |
|  | @CrossOrigin(origins = "\*") |
|  | public class StripeController { |
|  |  |
|  | @Autowired |
|  | PaymentService paymentService; |
|  |  |
|  | @PostMapping("/paymentintent") |
|  | public ResponseEntity<String> payment(@RequestBody PaymentIntentDto paymentIntentDto) throws StripeException { |
|  | PaymentIntent paymentIntent = paymentService.paymentIntent(paymentIntentDto); |
|  | String paymentStr = paymentIntent.toJson(); |
|  | return new ResponseEntity<String>(paymentStr, HttpStatus.OK); |
|  | } |
|  |  |
|  | @PostMapping("/confirm/{id}") |
|  | public ResponseEntity<String> confirm(@PathVariable("id") String id) throws StripeException { |
|  | PaymentIntent paymentIntent = paymentService.confirm(id); |
|  | String paymentStr = paymentIntent.toJson(); |
|  | return new ResponseEntity<String>(paymentStr, HttpStatus.OK); |
|  | } |
|  |  |
|  | @PostMapping("/cancel/{id}") |
|  | public ResponseEntity<String> cancel(@PathVariable("id") String id) throws StripeException { |
|  | PaymentIntent paymentIntent = paymentService.cancel(id); |
|  | String paymentStr = paymentIntent.toJson(); |
|  | return new ResponseEntity<String>(paymentStr, HttpStatus.OK); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/cavanosa/4f56145635963fd5419694cb0a860ce5/raw/cca051d7e3bdf038e194b908d752d74874b905e6/stripecontroller.java)[**stripecontroller.java**](https://gist.github.com/cavanosa/4f56145635963fd5419694cb0a860ce5#file-stripecontroller-java) hosted with  by [**GitHub**](https://github.com/)

En el primer método (payment) llamamos al servicio y creamos el paymentintent. Como respuesta devolvemos un string con el PaymentIntent convertido en JSON.

Los otros dos métodos, confirm y cancel recuperan un objeto PaymentIntent que será cancelado/confirmado. Ambos devuelven también un PaymentIntent convertido en JSON.

Y hasta aquí el backend. En la siguiente entrada llega el turno de Angular.