

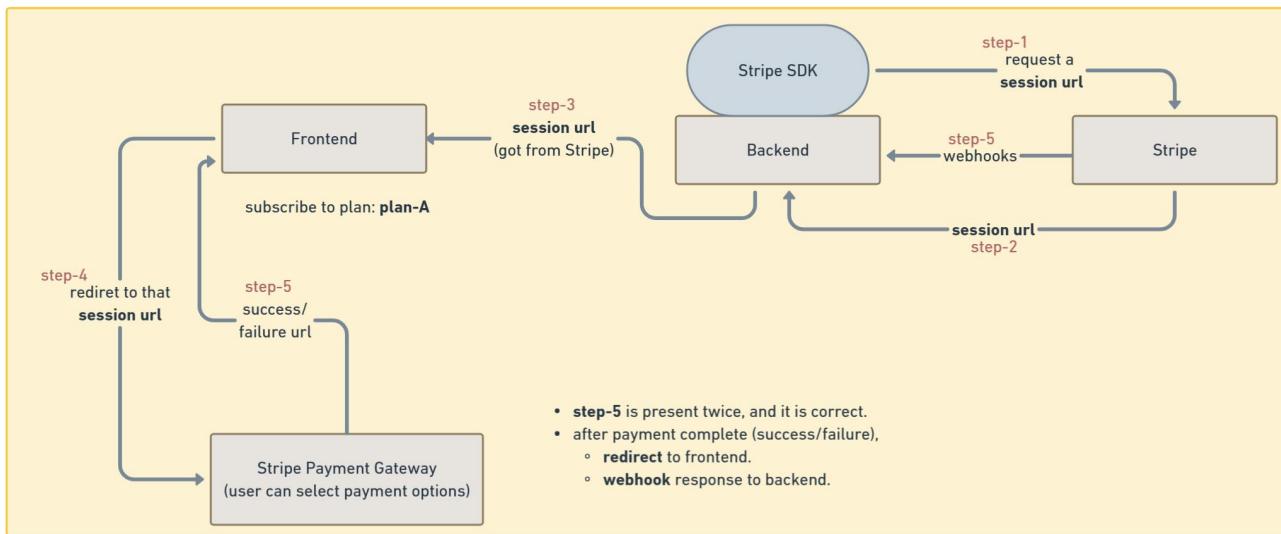
➤ **@PostConstruct**

- ~ It runs the method (which is being annotated with it) just after the bean is created.
- ~ **init()** method also does the same, but it was there in legacy XML config.

```
@PostConstruct  ↗ Alok Ranjan Joshi
public void init() {
    Stripe.apiKey = stripeSecretKey;
}
```

here, the method name can be anything, it is not bounded with **init()**, only the **@PostConstruct** matters here.

➤ Payment Integration Flow



➤ Why redirect to frontend and webhook for backend exists?

- ~ **Redirect** is to make the user know about the status, like success or failure.
But user can fake it by manually typing the url and will get a free subscription if webhook doesn't exist.
- ~ **Webhook** on the other hand, sends response to the backend about the status of payment, backend will verify it and stores in the database.
- There is a keyword called **volatile** in **Java**.
 - ~ `public static volatile String apiKey;` (Stripe class)
 - ~ **volatile** make sure that if one thread changes the variable, then other thread will immediately sees that.
 - ~ When you use **volatile**, all the thread will reference to variable present in main memory directly, not from the **thread local cache**.
- For stripe, you need to set the API key with **Stripe** class. [Stripe related]
 - ~ You will set that **apiKey** variable.

```

@PostConstruct  ↳ Alok Ranjan Joshi
public void init() {
    Stripe.apiKey = stripeSecretKey;
}
  
```

- As we need subscription based payment, like pro plan, normal plan etc, we need to create product in **stripe dashboard**. [Stripe related]

Product catalog

All products	Features	Coupons	Shipping rates	Tax rates
All				
2				
<input type="button" value="Created"/>	<input type="button" value="Status Active"/>	<input type="button" value="Clear filters"/>		
Name	Pricing	Created	Updated	
 Business Plan	\$1,499.00 USD ₹ Per month	Jan 18	Jan 18	
 Pro Plan	\$499.00 USD ₹ Per month	Jan 18	Jan 18	

- Now you need to make the **entity** for **Plan** as well. [Stripe Related]

```
@Column(unique = true)
String stripePriceId;

Integer maxProjects;
Integer maxTokensPerDay;
Integer maxPreviews; // max
Boolean unlimitedAi; // unli
```

- That **stripePriceId** is there inside the stripe dashboard.
- Its not the **Product_id**, its **Price_id** (both are different in Stripe Dashboard)
- And these should be directly appended by database, or you can create a REST end point that can only be accessed by the authorized people.
- One product (in our case Plan) should have same price, why there can be multiple prices under a single product in Stripe?

Product: Premium Plan

- |— Price 1: ₹499 / month
- |— Price 2: ₹4999 / year
- |— Price 3: \$9.99 / month (USD)

(you can think like this)

- One **product_id** can have **multiple price_id**.

active	max_previews	max_projects	max_tokens_per_day	status	unlimited_ai	id	name	stripe_price_id
[v]	1	3	10,000	[NULL]	[v]	1	Pro Plan	price_1SquHjJlllxid49U1UMNplA5
[v]	3	10	50,000	[NULL]	[v]	2	Business Plan	price_1SquKlllxid49UhMX0TiSU

- <https://docs.stripe.com/billing/quickstart> here you can checkout the Stripe integration docs.

Creating Checkout Session URL

```
var paramsBuilder = SessionCreateParams.builder()
    .addLineItem(
        element: SessionCreateParams.LineItem.builder()
            .setPrice(plan.getStripePriceId())
            .setQuantity(1L)
            .build()
    )
    .setMode(SessionCreateParams.Mode.SUBSCRIPTION)
    .setSubscriptionData(
        new SessionCreateParams.SubscriptionData.Builder()
            .setBillingMode(SessionCreateParams.SubscriptionData.BillingMode.builder()
                .setType(SessionCreateParams.SubscriptionData.BillingMode.Type.FLEXIBLE)
                .build()
            )
            .build()
    )
    .setSuccessUrl(frontendUrl + "?success=true&session_id={CHECKOUT_SESSION_ID}")
    .setCancelUrl(frontendUrl + "?cancel.html")
    .putMetadata("user_id", userId.toString())
    .putMetadata("plan_id", plan.getId().toString());
```

- ~ Creates a Stripe Checkout Session to start a payment flow
- ~ Adds one line item using a specific Stripe Price ID (not product) [**addLineItem**]
- ~ Sets the purchase type as a recurring subscription [**setMode**]
 - ~ Ensures Stripe handles automatic recurring billing
- ~ Uses flexible billing mode to allow future plan changes (upgrade/downgrade, proration) [**setSubscriptionData**]
- ~ Redirects the user to a Stripe-hosted checkout page
 - ~ On successful payment, redirects back to the frontend with the Checkout Session ID [**setSuccessUrl**]
 - ~ On cancellation, redirects the user back to the frontend cancel page [**setCancelUrl**]
- ~ Attaches metadata [**putMetadata**]
 - ~ Attaches internal user ID as metadata for payment tracking
 - ~ Attaches internal plan ID as metadata to identify which plan was purchased
 - ~ Allows backend to later verify payment and subscription via *webhooks*
- ~ Keeps payment handling PCI-compliant by using Stripe Checkout.
- ~ **Here I am creating a session *builder*, not *session*; if you see, I have not called the **build()** method at the end.**

```

Redirect → "Tell the user what happened"
Webhook → "Tell the system what actually happened"

```

- After that you need to create session

```

Session session = Session.create(paramsBuilder.build());
return new CheckoutResponse( checkoutUrl: session.getUrl());

```

- This **Session.create(...)** method makes an API call to stripe and get a **session** object.
- This **session** object contains the **url** that needs to be sent to the frontend.

- One catch is there here:

- Even if the same user is doing payment, then also inside Stripe, multiple **customers** will be created, because Stripe doesn't know about our customers.

<input type="checkbox"/>	Customer	Email	Primary payment method
<input type="checkbox"/>	Alok	user1@gmail.com	**** 4242
<input type="checkbox"/>	Alok	user1@gmail.com	**** 4242
<input type="checkbox"/>	Alok	user1@gmail.com	**** 4242

- Same gmail id (same user), but different customers got created inside Stripe.

- Now to make the Stripe knows if the user is same or not, you need to do the below.

```

String stripeCustomerId = user.getStripeCustomerId();
if(stripeCustomerId == null || stripeCustomerId.isEmpty()) {
    paramsBuilder.setCustomerEmail(user.getUsername());
} else {
    paramsBuilder.setCustomer(stripeCustomerId); // stripe customer id
}

Session session = Session.create(paramsBuilder.build()); // making API call
return new CheckoutResponse( checkoutUrl: session.getUrl());

```

- Now you can see, the email is fixed (if the user is already present), & you cannot edit the email id in Stripe payment page.

Pay with ⚡ link

Pay with amazon
Use your Amazon account

OR

Contact information

Email user1@gmail.com

Payment method

Card

Card information

- **But in database we don't have the customer id present; this will be handled in webhook event handling.**

- There are multiple events get triggered

Events

A Checkout Session was completed

Customer cus_ToZjJRYf6PwfrD subscribed to
price_1SquHjIlliixid49U1UMNpIA5

user1@gmail.com's payment for an invoice for
\$499.00 succeeded

user1@gmail.com's invoice for \$499.00 was
paid

A draft invoice for \$499.00 to
~~user1@gmail.com was finalized~~

- Here you can see the customer id and stripe id present (top 2nd event)
- So, you need to create a **webhook** that listens to this event and you'll get the customer id.
- And also, one webhook to see the payment status (success/failure)

- []

Now Frontend will get this checkout session URL;

Then Frontend will redirect to that URL;

Then after payment completion, Stripe will respond back to the Redirect URL to frontend & trigger Webhook to send response to the backend.

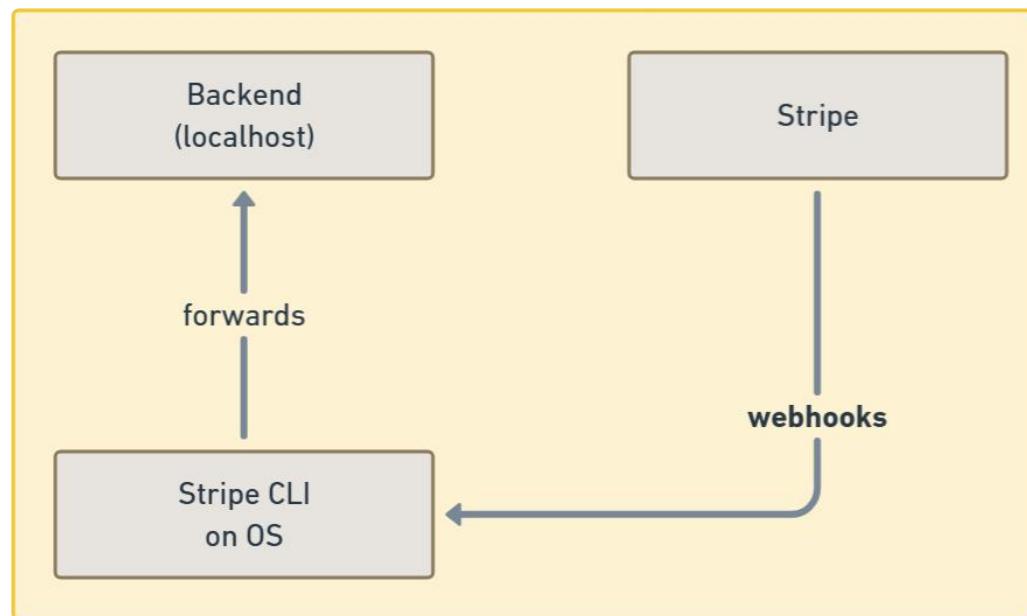
--- so we need to create the **webhooks** now ----

]]

- Stripe docs for java: <https://docs.stripe.com/api/errors?lang=java>
- F
- F
- F
- F

Webhooks

- As our backend is running on *localhost*, Stripe won't be able to send the webhook response to our backend, so we need something to listen to those and forward to localhost.
 - ↖ Stripe CLI does this. You need to configure this.
 - ↖ Stripe CLI docs: <https://docs.stripe.com/stripe-cli/use-cli>



- ↖ When you run the **stripe listen** command in CLI, you'll get a secret called *webhook signing secret*

```
PS C:\WINDOWS\System32> stripe listen --forward-to localhost:8080/webhooks/payment
A newer version of the Stripe CLI is available, please update to: v1.35.0
> Ready! You are using Stripe API Version [2025-12-15.clover]. Your webhook signing secret is
whsec_b5643f423774f00c8e858ad86d4e1e12a5a34f7dda4323cd149274e9bf1c5f6d (^C to quit)
```

- We discussed about **redirect** and **webhooks**, but still **webhooks** is not completely secure.
 - ↖ Because the webhook endpoint of the backend is a public url, and anyone can send request to that leading to get a free subscription.
 - ↖ So, stripe sends a **webhook signing secret** which is being validated at the backend that the request is coming from the Stripe or not.

- You can see in the inspector in the Stripe dashboard page about all the events that was triggered.

The screenshot shows the Stripe Dashboard's 'Events' tab. At the top, there's a search bar and a dropdown for 'LovableAI Alok'. Below the header, there are tabs for 'Overview', 'Webhooks', 'Events' (which is selected), 'Logs', 'Health', 'Inspector', 'Blueprints', and 'Shell'. There are also filters for 'Find event by ID...', 'Date', 'Status', 'Event type' (set to 'customer.*'), 'Resource', 'Payload type', and 'Res'. The main area shows a list of events. One event is highlighted: 'customer.subscription.created' for user1@gmail.com, which occurred on Jan 18, 2026, at 7:33:51 PM. The event details show 'user1@gmail.com subscribed to price_1SquHjlliixid49U1UMNplA5'.

- Create an end-point to listen the Stripe webhook.

- ↪ It'll always be a *post* end-point.

```
stripe listen --forward-to localhost:8080/api/v1/webhooks/payment
```

- ↪ Here you need to give the proper end-point that handle the webhooks.

```
@PostMapping("/webhooks/payment") ⚡ Alok Ranjan Joshi *
public ResponseEntity<String> handlePaymentWebhooks(
    @RequestBody String payload,
    @RequestHeader("Stripe-Signature") String sigHeader
) {
```

- ↪ Webhook handling controller.
- ↪ The header will be having the key **Stripe-Signature**, it contains the signature.

```
Event event = Webhook.constructEvent(payload, sigHeader, webhookSecret);

EventDataObjectDeserializer deserializer = event.getDataObjectDeserializer()
StripeObject stripeObject = null;

stripeObject = deserializer.getObject().get();

Map<String, String> metadata = new HashMap<>();
if(stripeObject instanceof Session session) {
    metadata = session.getMetadata();
}
```

- Now, when you make payment with the url that you get from the checkout response, you'll see the logs in the CLI (stripe listen ...etc

```
PS C:\WINDOWS\System32> stripe listen --forward-to localhost:8080/api/v1/webhooks/payment
A newer version of the Stripe CLI is available, please update to: v1.35.0
> Ready! You are using Stripe API Version [2025-12-15.clover]. Your webhook signing secret is whsec_b5643f423774fc
e858ad96d4c1e12a5a34f7dda4323cd149274e9bf1c5f6d (* to quit)
2026-01-31 17:56:24 --> charge.succeeded [evt_3SvdFnIIIXid49U0wRZA65Y]
2026-01-31 17:56:25 --> payment_method.attached [evt_1SvdFsIIIXid49UEXVI0xEU]
2026-01-31 17:56:25 --> customer.created [evt_1SvdFsIIIXid49UDtL3efZ8]
2026-01-31 17:56:25 --> checkout.session.completed [evt_1SvdFsIIIXid49UdWG6WTt8]
2026-01-31 17:56:25 --> customer.updated [evt_1SvdFsIIIXid49UfcigqExn]
2026-01-31 17:56:25 --> customer.subscription.created [evt_1SvdFsIIIXid49UamDePHce]
2026-01-31 17:56:25 --> payment_intent.succeeded [evt_3SvdhIIIXid49U0Mzb10mx]
2026-01-31 17:56:25 <- [200] POST http://localhost:8080/api/v1/webhooks/payment [evt_1SvdFsIIIXid49UdWG6WTt8]
2026-01-31 17:56:25 --> payment_intent.created [evt_3SvdFnIIIXid49U0jeuvCNJ]
2026-01-31 17:56:25 <- [200] POST http://localhost:8080/api/v1/webhooks/payment [evt_1SvdFsIIIXid49UamDePHce]
2026-01-31 17:56:25 --> invoice.created [evt_1SvdFsIIIXid49UEXVI0xEU]
2026-01-31 17:56:25 <- [200] POST http://localhost:8080/api/v1/webhooks/payment [evt_1SvdFsIIIXid49UEXVI0xEU]
2026-01-31 17:56:25 --> invoice.finalized [evt_1SvdFsIIIXid49U1JmK5tn0]
2026-01-31 17:56:25 --> invoice.paid [evt_1SvdFsIIIXid49U1JmK5tn0]
2026-01-31 17:56:25 --> invoice.payment_succeeded [evt_1SvdFtIIIXid49UDIEvdZfq]
2026-01-31 17:56:26 <- [200] POST http://localhost:8080/api/v1/webhooks/payment [evt_1SvdFsIIIXid49UFcigqExn]
2026-01-31 17:56:26 <- [200] POST http://localhost:8080/api/v1/webhooks/payment [evt_1SvdFsIIIXid49UDtL3efZ8]
2026-01-31 17:56:26 <- [200] POST http://localhost:8080/api/v1/webhooks/payment [evt_3SvdFnIIIXid49U0jeuvCNJ]
2026-01-31 17:56:26 <- [200] POST http://localhost:8080/api/v1/webhooks/payment [evt_3SvdFnIIIXid49U0Mz670mx]
2026-01-31 17:56:26 <- [200] POST http://localhost:8080/api/v1/webhooks/payment [evt_1SvdFsIIIXid49Uceh5KCGe]
2026-01-31 17:56:26 <- [200] POST http://localhost:8080/api/v1/webhooks/payment [evt_1SvdFtIIIXid49UGSJuNdaP]
2026-01-31 17:56:26 <- [200] POST http://localhost:8080/api/v1/webhooks/payment [evt_1SvdFtIIIXid49UDIEvdZfq]
2026-01-31 17:56:26 <- [200] POST http://localhost:8080/api/v1/webhooks/payment [evt_1SvdFsIIIXid49U1JmK5tn0]
2026-01-31 17:56:27 <- [200] POST http://localhost:8080/api/v1/webhooks/payment [evt_3SvdFnIIIXid49U0wRZA65Y]
```

- These all are events, *charge.succeeded*, *payment_method.attached*etc etc.
- And for these events, the webhook event handler endpoint will be triggered **once per event**. (here you can see all the POST requests)

```
@RequestBody String payload, payload: "{\n    \"id\": \"evt_1SvdQ6IIIXid49URxQBzE8y\", \n    \"object\": \"event\"\n@RequestHeader(\"Stripe-Signature\") String sigHeader sigHeader: "t=1769863028,v=1a57ec8fc3d0b5f6fa25004"
```

- These are the payload and signature header values.

```
event = {Event@16583} "<com.stripe.model.Event>"\n\n  account = null\n  > apiVersion = "2025-12-15.clover"\n  context = null\n  > created = {Long@16588} 1769863028\n  > data = {EventData@16589} "<com.stripe.model.EventData>"\n  > id = "evt_1SvdQGIIIXid49UTLdQj1YK"\n  > livemode = {Boolean@16591} false\n  > object = "event"\n  > pendingWebhooks = {Long@16593} 2\n  > request = {EventRequest@16594} "<com.stripe.model.EventRequest>"\n  > type = "checkout.session.completed"\n  > responseGetter = {LiveStripeResponseGetter@16595}\n  > lastResponse = {StripeResponse@16596}\n  rawJsonObject = null
```

- This is the **event** object that was created using the *secret key*, *signature header* & *payload*. You can see **event type** is also there.
- After then you create **EventDataObjectDeserializer** object using that **event object**,

- ❖ It contains a key “**object**” which contains an object of type **Session** (*it extends StripeObject class*)

```
deserializer = {EventDataObjectDeserializer@16625}
① apiVersion = "2025-12-15.clover"
① eventType = "checkout.session.completed"
① rawJsonObject = {JsonObject@16626} {"id":"cs_test_a1ps3g5GRx"
① object = {Session@16627} <com.stripe.model.checkout.Session@
```

- ❖ That **Session** object contains all the details like user's registered email in Stripe, the **metadata** that was being attached while creating the session (checkout session)

```
customerEmail = "user1@gmail.com"
```

```
└─ metadata = {Linked
  > └─ "plan_id" -> "1"
  > └─ "user_id" -> "1"
```

➤ F

➤ Events of Webhooks

- ~ checkout.session.completed [
 - ❖ This event occurs only once in our case as it is recurring.
 - ❖ User will only go to checkout page once and set the auto-pay kind of thing.
 - ❖ Then every interval (month/year) stripe will try to deduct by itself.
- ~ invoice.paid
 - ❖ This event is triggered when the payment is done successfully.

```
public void handleWebhookEvent(String type, StripeObject stripeObject, Map<String, String> metadata) {  
    log.info("type = {}", type);  
  
    switch (type) {  
        /// one-time, on checkout completed  
        case "checkout.session.completed" -> handleCheckoutSessionCompleted(session: (Session) stripeObject, metadata);  
        /// when user cancels, upgrades or any updates (----- cancel might not trigger this -----)  
        case "customer.subscription.updated" -> handleCustomerSubscriptionUpdated(subscription: (Subscription) stripeObject);  
        /// when subscription ends  
        case "customer.subscription.deleted" -> handleCustomerSubscriptionDeleted(subscription: (Subscription) stripeObject);  
        /// when invoice is paid (due payment or subscription auto-pay is paid successfully)  
        case "invoice.paid" -> handleInvoicePaid(invoice: (Invoice) stripeObject);  
        /// when invoice is not paid, mark as PAST_DUE  
        case "invoice.payment_failed" -> handleInvoicePaymentFailed(invoice: (Invoice) stripeObject);  
        default -> log.debug("Ignoring the event: {}", type);  
    }  
}
```

-
- ~ This is the webhook handler method where these many events are being handled.
[check the comments for their purpose]
 - ~ In Stripe, **StripeObject** is the parent class of all Stripe objects.

➤ handleCheckoutSessionCompleted(session, metadata)

```
private void handleCheckoutSessionCompleted(
    Session session,
    Map<String, String> metadata) {
}
```

- ❖ On, **checkout.session.completed**, a new subscription will be created, so here we'll set the customer id in our database.
- ❖ It contains all like subscription creation, invoice paid, customer updation ..etc.
- ❖

Subscriptions

Subscriptions										+ Create test subscription	...
Subscriptions		Test clocks		Migrations							
Active		Scheduled		Canceled		Simulated		All			
Price	Simulated	Created date	Status	Active	Customer ID	More filters	Clear filters	Export	Analyze	Edit columns	...
Customer	Status	Customer name	Customer description	Billing	Tax calculation	Product	Created	Average monthly total	Average yearly total
user1@gmail.com	Active	Alok		Auto	None	Pro Plan	Jan 31, 7:15 PM	\$499.00 USD / month	\$5,988.00 USD / year
user1@gmail.com	Active	Alok		Auto	None	Pro Plan	Jan 31, 6:07 PM	\$499.00 USD / month	\$5,988.00 USD / year
user1@gmail.com	Active	Alok		Auto	None	Pro Plan	Jan 31, 5:56 PM	\$499.00 USD / month	\$5,988.00 USD / year

- ❖ These are all the subscriptions.
- ❖ Each subscription is having one **unique id**.

Customers

All								Remaining balances		
Email	Name	Created date	Type	More filters						
<input type="checkbox"/>	Customer	Email	Primary payment method	Country	Created	Total spend	Payments	Ref		
<input type="checkbox"/>	Alok	user1@gmail.com	**** 4242	India	Jan 31, 7:15 PM	\$499.00 USD	1	\$0.00		
<input type="checkbox"/>	Alok	user1@gmail.com	**** 4242	India	Jan 31, 6:07 PM	\$499.00 USD	1	\$0.00		
<input type="checkbox"/>	Alok	user1@gmail.com	**** 4242	India	Jan 31, 5:56 PM	\$499.00 USD	1	\$0.00		

- ❖ These all are customers (redundant till now, because customer id is not stored in DB yet).
- ❖ Each customer is having one **unique id**.
- ❖

➤ F

```
> F
paymentProcessor      : type = customer.subscription.deleted
thFilter              : /api/v1/webhooks/payment
paymentProcessor      : type = invoice.updated
thFilter              : /api/v1/webhooks/payment
thFilter              : /api/v1/webhooks/payment
> paymentProcessor   : type = refund.created
> F
> F
> F
> F
> F
> F
>
```