

My imaginary use case

I have always wanted to start my own business and what better place to begin than commercializing one of my girlfriends most annoying habits - making face scrubs out of used coffee grounds. I am based in the US and want to rapidly test the market for our homemade scrubs. I studied literature and have no prior knowledge of payments. I recently completed a python course online and now believe I am a bona fide software developer so making a website to accept payments for my fabulous products should be a piece of cake, right?

Accept a payment doc page

I sit down with my coffee, roll up my sleeves and open the [accept a payment](#) page on Stripe. This is the first text I see.

Collecting payments on your website consists of creating an object to track a payment, collecting card information, and submitting the payment to Stripe for processing.

Stripe uses this payment object, called a `PaymentIntent`, to track and handle all the states of the payment until it's completed—even when the bank requires customer intervention, like two-factor authentication.

Immediately my bona fide developer ego takes a slight hit as *conceptually* I am unsure what is meant by “creating an object to track a payment”. There are no links to explanations or additional context in this opening section. When I scroll for a while I find that [step 2](#) has a clickable link for `PaymentIntent`, I click it.

2 Create a PaymentIntent

Stripe uses a `PaymentIntent` object

It brings me to an API reference page where I see “The `PaymentIntent` object” and this brings me some relief. Glancing at this object helps me understand what this opening section of text means. I see an id, an amount, a status, currency etc. and it is an aha moment, I can now visualize how the different stages of a payment can be tracked by an object such as this.

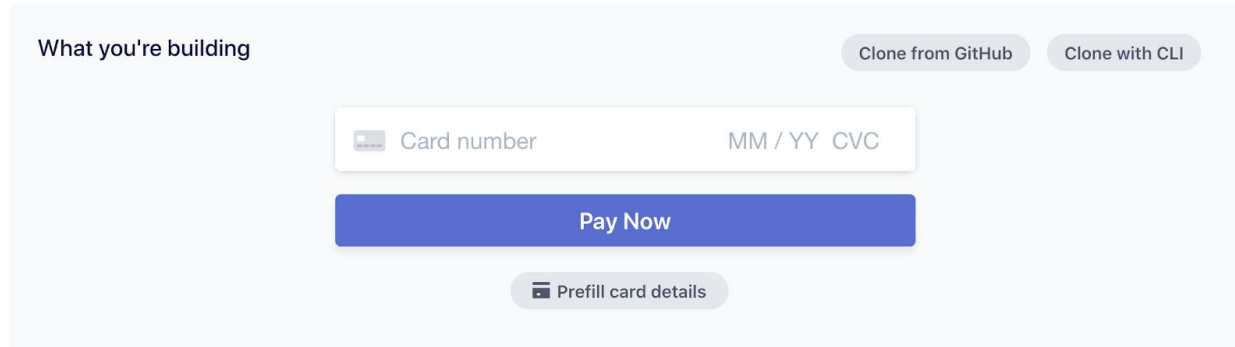
THE PAYMENTINTENT OBJECT

```
{
  "id": "pi_1Gar2oBJd1g7CJPip78SrGPZ",
  "object": "payment_intent",
  "amount": 1000,
  "amount_capturable": 0,
  "amount_received": 0,
  "application": null,
  "application_fee_amount": null,
  "canceled_at": null,
  "cancellation_reason": null,
  "capture_method": "automatic",
  "charges": {
    "object": "list",
    "data": [],
    "has_more": false,
    "url": "/v1/charges?payment_intent=pi_1Gar2oBJd1g7CJPip78SrGPZ"
  },
}
```

Feedback for PI team

If we are linking off to the PaymentIntents section in the API docs, could we do this when we first introduce PaymentIntents instead of step two? Inpatient developers may not want to wait until later to get more context on a new and crucial concept.

The "What you're building" box/section



What you're building

Clone from GitHub Clone with CLI

Card number MM / YY CVC

Pay Now

Prefill card details

Initially, the visual is a welcome relief after searching for context on what PaymentIntent objects look like and how they can help me sell face scrubs. There is a nice moment of thinking "oh cool I get to embed one of these on my website". Selling my face scrubs online starts to feel a little bit more real.

Quickly however I come to find the "What you're building" statement to be a little oversimplified. Step 1 and 2 below refer to setting up Stripe (server side) and creating a PaymentIntent (server side) respectively, neither of which seem *directly* involved in building/rendering the credit card details collection box. When I scroll through to step 3 I see "Collect card details" and this makes me associate step 3 with the box that I am told I am building. I scroll back to the top of the page again and re-read the first 3 lines.

Collecting payments on your website consists of creating an object to track a payment, collecting card information, and submitting the payment to Stripe for processing.

Ok, so now I am slowly starting to form a mental model of the overall flow of the integration I am building. I see that collecting payments consists of creating an object to track a payment, collecting card details (via the box I have been told I am building) and submitting the payment to Stripe (I have no context on this part at this point in time, I start to scroll again and eventually see it is step 4). I jot down three points on my notepad to help me visualize the mental model that is forming:

1. Create a PaymentIntent object
2. Collect card details
3. Submit the payment to Stripe

This felt like another mini a-ha moment as I got a mini kick of pride thinking that the docs are starting to make sense to me. When the dopamine kick wears off I am left with a feeling that I wish the docs had an illustrative overview of the flow of the whole integration. This lack of higher level illustration may create unnecessary cognitive processing time for impatient developers.

Feedback for PI team

Illustrating only the card details collection form and not the entire integration flow in the opening section may leave room for creating unnecessary cognitive processing lift for impatient developers.

A simple and succinct illustration could save a lot of intense doc reading and pondering. Why not make it obvious from the get go what the integration looks like and how it flows through different states?

1 Set up Stripe

No issues here, the only question I have is does such a simple task justify a full step onto itself? Scrolling down, steps 2-4 seem to be very bulky compared to this very simple step 1 task. Perhaps the idea was to give the developer an “easy win” at the beginning?

2 Create a PaymentIntent (server-side)

I click the PaymentIntent API hyperlink, reading this really let's me know what paymentIntents is all about - “exactly one” for each order or customer session. Makes sense. The visuals in the multiple statuses hyperlink really drive this home and make it very clear why it has been designed this way and how I am supposed to use it. As a European I can see how this would be compatible with 3D secure and SCA, simple but effective, I like it.

I now fully “get” PaymentIntent objects and that they can handle payment status, let's create one! The next sentence says create a paymentIntent on my server with the amount and currency and the tip about always doing this to prevent malicious customers *fee/s* cool. I can envision myself telling my girlfriend ‘I made sure to create the payment object on the server, not the client, to ensure we are never hacked. Pretty, cool, eh?’ and her hopefully being impressed and not rolling her eyes...

Next my eyes scan the code block for server.py - the first thing I read is about setting my secret key. This is the first time “secret key” is mentioned, I have no idea what Stripe is talking about. I scroll back up to make sure I did not miss previous mentions of secret keys - I did not. I copy and paste the dashboard link in the code block into my browser (would a hyperlink here be an easy win?), log in and see that I have two keys, publishable and secret, and there is no context on either. I eventually decided to just roll with it and follow the code examples and hope that things become clear later, though I would rather know now.

Below the code block I am introduced to another new concept, a client secret. However, thankfully this concept has an intro when I hover and a hyperlink for more info. When I hover I am told that the client secret is a unique key...this is the third key I have read about in quick succession now without having keys and their uses contextualized. I click the hyperlink to the API ref, and at the top of the page I see this:

 **client_secret** string RETRIEVABLE WITH PUBLISHABLE KEY

I am confused, I click back to the code block as I thought I recalled the secret key being mentioned there, not a publishable key, I was correct.

```
1 # Set your secret key. Remember to switch to your live secret key in production!
2 # See your keys here: https://dashboard.stripe.com/account/apikeys
3 stripe.api_key = 'sk_test_Y0ENa4GPK1AA1wxrvHTQ8IRf00K8VBCgGc'
```

I am now potentially confused. Let's summarize what just happened:

1. The sample code told me to set my *secret key* (I was not told why I have to do this or what this is)
2. The next line of text says a client secret will be returned, I am told this is a *unique key*
3. The API ref tells me that the client secret is retrievable with a *publishable key*

What is going on? This step is called Create a PaymentIntent, but I am lost in a sea of mentions of keys and I have no context on how they relate to creating this PaymentIntent.

I google "how does Stripe payment flow work". I eventually find a non-Stripe website that has an [article](#) called "introduction to the Stripe API for java". I can deduce that this article is not for the paymentIntents API as the flow is different, and I am using python not java, however the description of the flow steps e.g. "Our back-end contacts Stripe with the token, the amount, and the secret API key" helps me visualize a little bit how the keys are used.

This non-stripe website article helped drive home for me that *keys are used to communicate with Stripe*, it then clicks with me that all this key jazz is likely about authentication and tying myself and the purchases I wish to receive to my stripe dashboard where the keys are located. I come back to the docs and look at the code block once again, I conclude the secret key must be used to actually create the payment intent and by looking at the API ref I conclude that the secret key is returned. I studied literature, and so payments security etc. is foreign to me. But I am starting to get the *sense* that all this "key" and "secret" talk is about security and is actually very well thought out, perhaps just not explicitly explained / contextualized in this "accept a payment" page. I also think maybe alternatively I am just dumb for not knowing this. Despite figuring this out myself with some googling, my bona-fide developer ego takes a hit.

I feel like things make sense again, except the line "is retrievable with publishable key", I don't know what is meant here and am unsure why I would want to retrieve this. I read on. "The client secret can be used to complete a payment from your frontend." - aha! This is the first time I have formed a semi complete mental model of the entire payment flow:

1. Create PI on backend with secret key
2. Get client secret back for use later confirming a payment on the frontend
3. Complete payment on frontend with client secret (in conjunction with publishable key?)

Reading the bullet points above would have saved me an hour or two of googling and feeling confused.

Once my mind settles a little - I am now left with another big question - why am I completing the payment on the frontend? I am building this little store to see if anyone will buy my girlfriends coffee grounds based face scrubs (did I mention they are organic and vegan *and* reverse aging?) - what if people lose internet connection temporarily during purchase? Will I totally lose the payment??

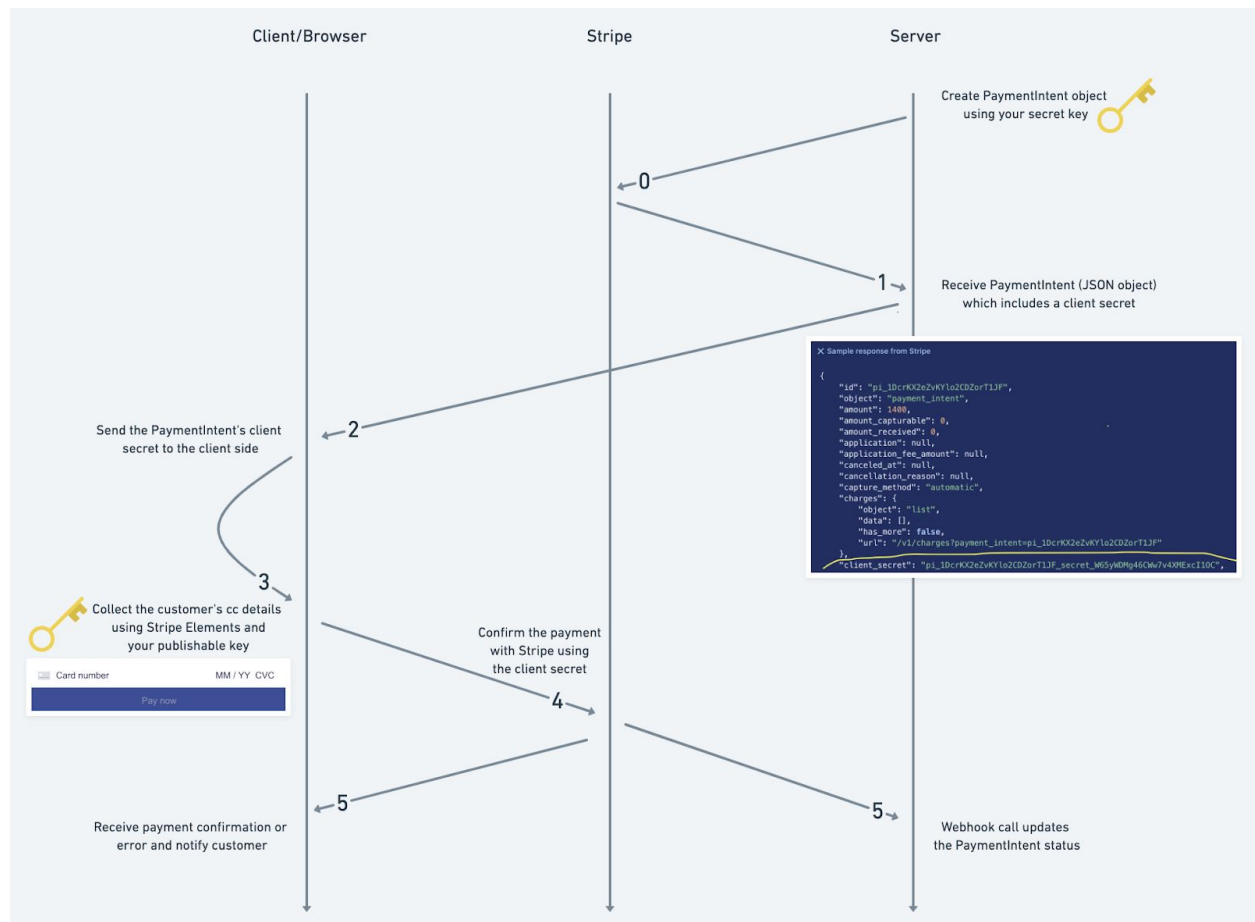
I can't see anything on this page about confirming the payment on the server, so I get the sense that this is how Stripe wants me to do it. It feels slightly "off" to have no context or pros/cons about why I am being told to confirm on the client.. Moving on, I am going for server-side rendering and really enjoy this snippet of text and code block. I like that I can embed my secret in the HTML output of the checkout page upon

rendering via flask. This *feels* cool, like a secret hidden language I have with Stripe that elegantly enables the unknowing end-user of my site to buy our fantastic face scrubs with ease.

Feedback for PI team

1. As mentioned previously, an illustration early on in the docs giving an overview of the whole integration flow and the different steps e.g. create PI with secret key, get client secret back etc. could save a lot of mental processing time.

The example image below won't be winning any design awards but it would help the impatient developer visualize what happens at each step. Something like this would have saved me a few hours.



2. Context on why we have the two keys and what they do when they are first introduced would really help.
3. I am sure we have good reasons for confirming on the client side (maybe it helps with radar? Maybe it is stickier as we have code embedded in their frontend too?), but I imagine a lot of users would prefer server side confirmation and so some context as to why we promote this flow, pros/cons and why we should not be worried about client connectivity issues etc. may calm some fears and build more confidence in the integration/Stripe.
4. Nit: making the dashboard link in the code block a hyperlink would save a copy and paste.

3 Collect card details (client side)

"You're ready to collect card information" - I am genuinely excited. I hit the elements doc page as it is really, really nice. Scrolling down and seeing the "result - HTML - CSS - JavaScript" with code snippets in each makes me think this is what Stripe is about, brilliant docs that do the heavy lifting for me and semi floor me with convenience/thoughtfulness/lovely design. I also get clarification on what the publishable key is used for i.e. creating an instance of the Stripe object.

This line "Elements is a set of prebuilt UI components for collecting and validating card number, ZIP code, and expiration date" creates a nice sense of wonder, as I never really thought about where the card details go, and now I know I don't have to either as Stripe is taking care of it. Having read the opening section here along with the elements docs, I now have the mental association of the "what you're building" box from the top of the docs with "elements".

The HTTPS mentions adds a nice sense of "Stripe will help me be secure, I can trust them". It has been a long time since I did any coding besides messing with microcontrollers so having the option to go old school and not use react is greatly appreciated.

Create an instance of Stripe -> create an instance of elements -> create an instance of an element, the latter two steps sound so similar it took a moment to register that a card is just one type of element from elements. It's not a big deal but I wonder is there any wording here that could be added to avoid any potential for confusion reading these successive steps. The fact that elements validate as the end user input is typed is great, this could save a lot of my customers time. A final wow moment is reading that postal code fields are defined dynamically based on the card, cool!

Feedback for PI team

This whole step 3 is fantastic. Super clean and easy flowing and makes me very impressed with Stripe. Docs are really great. The only tiny nit is the language around the successive steps of creating an instance of elements -> creating an instance of an element language taking a second to register.

Feature suggestion: could we create an element for a cart? I know carts are common in checkout flows, however, it seems like a lot of work and I just want to rapidly test if there is a market for these face scrubs so I won't build one now, but I feel this would be greatly appreciated by a lot of impatient developers.

4 Submit the payment to Stripe (client side)

This text "Rather than sending the entire PaymentIntent object to the client, use its client secret from step 2. This is different from your API keys that authenticate Stripe API requests." is great and makes so much sense - why not have something as explicit as this when we first introduce keys and the client secret? At this point I feel like I have a lot of clarity as to how the integration works and what happens at each step, but again I could have really benefited from having this picture in my head up front. The tips on how to handle the client secret are appreciated as is the option not to use react.

This section is smooth and I feel like I am near the finish line until I see the following comment:

```
if (result.paymentIntent.status === 'succeeded') {  
  // Show a success message to your customer  
  // There's a risk of the customer closing the window before callback  
  // execution. Set up a webhook or plugin to listen for the  
  // payment_intent.succeeded event that handles any business critical  
  // post-payment actions.
```


Hark! I recall having FUD about client side confirmation earlier because of fears of connectivity issues and here my fears are validated. It would have been great to know this could be a drawback to this approach early on rather than now when I am close to the end of the integration. It also feels a bit raw to be told now that I am going to have to use webhooks to accommodate for this flow and “business critical post-payment actions”. I have never used webhooks before so this is a little startling to be hit with when I thought was the end of the integration.





Feedback for PI team

Integration wise, this step was very smooth. However, I did not appreciate having something as critical to the success of my business (i.e. webhooks to ensure receiving all of my orders reliably) being brought up 1) near the end of the integration and 2) subtly in a comment of a code snippet. This gave me the feeling that Stripe takes this kind of thing slightly for granted, which is a total disconnect for me, as I *never* want to have a customer place an order which won’t be fulfilled. That would be a horrific experience for my users that would erode trust in my company and spawn scathing, eventual-business-killing reviews online.

If this client side confirmation approach is the best for me and is worth the webhook hassle, I should be educated at the beginning of the integration as to why. I should also be educated about the trade-offs of client vs. server confirmation up front e.g. the former requires webhooks. The way it is currently, I feel like I have been guided through an integration that Stripe obviously wants but could be a lot of hassle for me as my business grows. This leaves a little bit of a bad taste in my mouth.

5 Test the integration

The instantaneous feedback here is wonderful, reminds me of duolingo or the likes, I feel delighted with myself lighting up the green tick boxes. The modal demo for card auth in Europe etc. is really cool, even though I don’t need it now as I am focused on the US, it’s cool to know if my coffee scrub business takes off that I am built for scale.

TEST SCENARIO	TEST CARD NUMBER
> Your integration handles payments that don’t require authentication ✔ All tests completed	4242 4242 4242 4242 
> Your integration handles payments that require authentication ✔ All tests completed	4000 0025 0000 3155 
> Your integration handles card declines codes like <code>insufficient_funds</code> ✔ All tests completed	4000 0000 0000 9995 
 Check payments 3 test payments found.	

Run the tests above to check the most recent test payments on your currently logged in Stripe account, [Séamus testing, Inc.](#)

Feedback for PI team

This is great and fun! Wonderful validation of the integration.

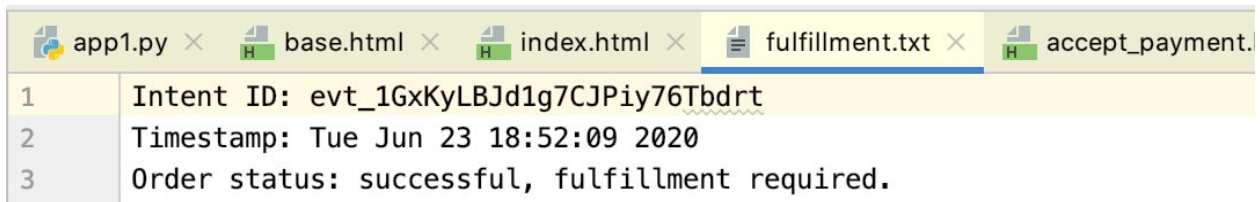
Optional Handle post-payment events

As my initial goal is just to test demand for these face scrubs, I don't anticipate huge volumes of orders, none the less it seems like checking if payments succeeded manually is not a sustainable activity. In this regard, I feel forced into webhooks. Also, I get a sense of demoralization as I felt like I was at the end of the integration, but now face a big unknown. Before moving on to look at the code for integrating / testing webhooks, I see this little bombshell tucked away in an info box...

i To closely control when funds move, you can [manually authorize the payment](#) and finalize it on your server. This integration is significantly different and has limitations.

I click the link....BAM! "Finalize payments on the server". I had no idea this existed until now. If I followed this doc from the beginning would I not have to deal with pesky webhooks? So many questions, too little time. I am way past the point of no return integration style wise i.e. it's complete (bar webhooks). Finding out that docs for this approach exists right at the very end did not feel good. I see a set of limitations here but I see no pros/benefits? Surely this way means no webhooks are required and that bad internet would not lose me valuable face scrub sales? Those are surely benefits worth mentioning? It's too late now anyway, I continue..

I hit "build a custom webhook" and start browsing those docs. I do a little browsing on google to understand and conceptualize webhooks better. Following the docs, the CLI steps are very straightforward and logging successful and failed orders in a .txt file soon became automagic!!



```
app1.py × base.html × index.html × fulfillment.txt × accept_payment.  
1 Intent ID: evt_1GxKyLBJd1g7CJPiy76TbdrT  
2 Timestamp: Tue Jun 23 18:52:09 2020  
3 Order status: successful, fulfillment required.
```

Successful order log using webhook CLI tool

The ease with which the CLI tool enabled me to test webhooks did reduce my anxiety towards them somewhat i.e. I thought there would be much more work here, however, I still have apprehensions about actually deploying them in the future.

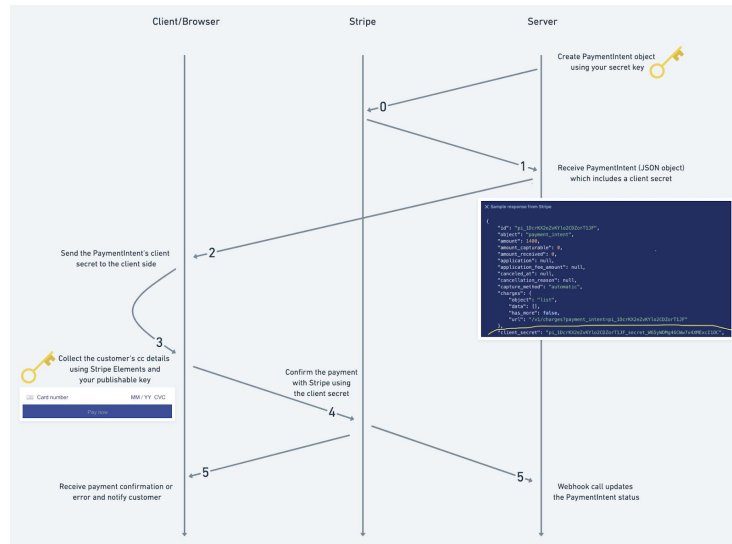
Feedback for PI team

Could we tell people about the server side integration at the beginning of the docs and give people a fair benefit/trade off summary up front between it and this client side confirmation approach? This may help avoid leaving a bad taste in the mouth of anyone impatient developers that hate webhooks and would prefer having server side confirmation.

Finally, the CLI testing tool is very impressive.

Friction log summary

1. I spent more time comprehending than I did coding. Explanations of/context on key concepts were not always present when the key concepts were first introduced e.g. PaymentIntents, client secret, secret key etc.
2. Never ending text and no block diagrams, images etc. does not enable visual learners like myself. An overview image at the beginning would have saved me hours.



3. I felt slightly "lead" into using client side confirmation. Being forced into a "listen" state feels like a loss of control. Having a link to the server side confirmation approach at the very end and the docs only stating its limitations and not its benefits left a slightly bad taste in my mouth.
4. Elements for cart/basket functionality would have been sweet.