Exercise 8: Stripe API

The exercises so far have used imaginary, fairly simple APIs. Let's now take a piece of an actual API and create an OAS file for it. Expect this exercise to take some time, at least a couple of hours.

Stripe is a company that has an API for credit card transactions. Online companies often use it as a way to charge their customers. Customers will enter credit card information at a company's website, and then the company's servers will make calls to the Stripe API to make those transactions happen. This way, the companies do not need to set up relationships with credit card companies. Stripe is used by a very large number of companies and handles billions of dollars every year.

Stripe pioneered a form of API documentation where the page is divided into three vertical slices: a navigation bar, the documentation, and sample code. The sample code shows multiple languages as tabs that you can switch between. For this exercise, you will only use the curl tab.

There's no reason to think that Stripe uses OAS, but in this exercise, imagine that they do. It would take weeks to create an OAS file for the entire Swagger API, so we are going to just use one piece of it, which is the Usage Records section resource. The Usage Records API requests are used to send customer usage and metrics to Stripe. I chose this because it's smaller than most of the resources, but it will still allow you to make a real-world example.

The documentation for this endpoint is at: https://stripe.com/docs/api/usage_records?lang=curl. Bring that up in a webpage so you can refer to it. Also bring up other OAS files that you can reference.

Important! This exercise is based on the Stripe website, which could change at any time if they change their API. Please notify me if you see something different, or if the links no longer work, etc.

General information

Open up the Swagger editor and from the **File** menu, clear the editor.

1. Start with the Swagger version of 2.0.
2. Now add the **info** section. You will need to find the **version**. Open https://stripe.com/docs/api/versioning?lang=curl and notice the current version. As of the writing of this exercise, that version is **2019-02-19**.
3. For the title, say "Stripe Usage Records".
4. For the description, take the description from https://stripe.com/docs/api/usage_records?lang=curl. Note that **metered billing** is a link to https://stripe.com/docs/billing/subscriptions/metered-billing. Use this Markdown to indicate the link, placing it where the text says "metered billing":

```
[metered billing](https://stripe.com/docs/billing/subscriptions/metered-billing)
```

5. Now you are done with the **info** section. Next, add the **host**, **basePath**, and **scheme**. To figure that out, look at one of the Usage Records endpoints: https://stripe.com/docs/api/usage_records/create?lang=curl. To the right, under the Curl tab, you will see an example request. After the word "curl", you can see the URL, which contains the **host**, **basePath**, and **scheme**. As of the writing of this exercise, the host is **api.stripe.com**, the **basePath** is **/v1**, and the **schemes** are just one item: **https**
6. Finally, add **consumes** and **produces**. If you look at the introduction (https://stripe.com/docs/api?lang=curl), you will see that the API accepts form-encoded data and returns JSON. This means that **consumes** should have one item of **application/x-www-form-urlencoded** and **produces** should have one item of **application/json**

Although you will get an error because you don't have path information, you should be able to see the title and description with the link, as well as the base URL, which should mention the Stripe documentation. If you are stuck, you can look at my solution so far at: http://sdkbridge.com/swagger/stripe1.yaml

Security

If you read the Authentication section (https://stripe.com/docs/api/authentication?lang=curl), you will find that authentication is performed using BasicAuth. So you will need to create a security definition which you can call **basicAuth** that has a type of **basic**.

Add that to the bottom of your OAS file.

Finally add a security section under the produces section so that we use the basicAuth definition:

```
security:
  - basicAuth: []
```

The Usage Record object

The first part of the Usage Record section of the documentation is the usage record object (https://stripe.com/docs/api/usage_records/object?lang=curl). We're going to use some advanced OAS features that I didn't cover to make it match the Stripe API better. To add it to your OAS file:

1. Add a **definitions** section above the **securityDefinitions** key.
2. Within that, add an object called **usageRecord**.
3. Within that, add a **properties** key, indented.
4. Look at the documentation at the link above, and you will see there are 6 properties to add: **id**, **object**, **livemode**, **quantity**, **subscription_item**, **and timestamp**. Add an **id** property of type **string**, and for the **description**, copy the text from the documentation. ("Unique identifier for the object.")
5. Add an **object** property. For type, use string. Again, take the description text from the documentation. Note that object always has a value of "usage record". There are two ways you can handle this. You can simply add the following sentence at the end of the description: value

is "usage_record". Or you can add an “enumeration”, which lists the valid values it can have. In this case, there is only one valid value, which is “usage record”.

```
      object:
        type: string
        description: String representing the object's type.
Objects of the same type share the same value.
        enum:
          - usage_record
```

6. Add a **livemode** property of type boolean. Take the description text from the documentation. To make the “true” and “false” be monospace font, put them in backticks. For example:

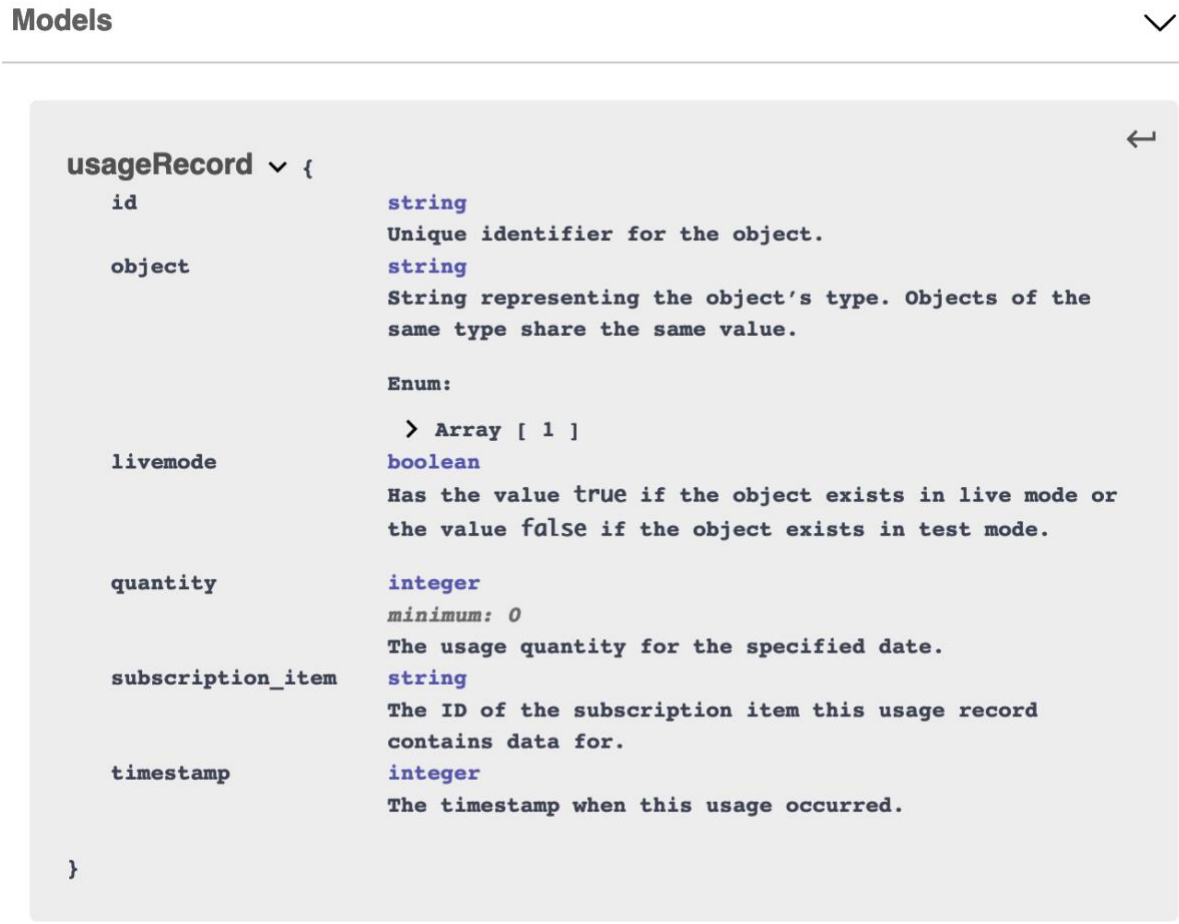
```
description: Has the value `true` if the object…
```

7. Add a **quantity** property of type integer. Take the description text from the documentation. To make it so that it can only be zero or higher, add a **minimum** key, like this:

```
quantity:
  type: integer
  description: The usage quantity for the specified date.
  minimum: 0
```

8. Lastly, add **subscription_item** and **timestamp**, following the same pattern as above. Note that a type of **timestamp** is not considered valid in OAS 2.0, so you will need to use **integer** instead. For Stripe, timestamp is an integer that encodes a date and time.

On the right side of the editor, you will still see an error because we not put in the **paths** key yet. But if you scroll down to the bottom, you will see your model, which should look like this:



Note that I clicked on **> Array [1]** to see the **[usage record]**.

If you are stuck, you can look at my solution at: http://sdkbridge.com/swagger/stripe2.yaml

Create a usage record

If you keep scrolling down, you will see the first API request, which is to create a usage record. (https://stripe.com/docs/api/usage_records/create?lang=curl). In this section, you will figure out information from the curl sample section of the documentation. curl is a command-line tool used to make HTTP requests. Follow these steps to add your first path:

1. Under the **security** section, add a **paths** key, like you have done in previous exercises.
2. Look at the right side, under the **curl** tab. Note that the URL is: **https://api.stripe.com/v1/subscription_items/si_DAyhPiOTjrdfTv/usage_records.** In this case, the **api.stripe.com** is the host, the **v1** is the base URL, and the path is **subscription_items/si_DAyhPiOTjrdfTv/usage_records**. However, **si_DAyhPiOTjrdfTv** is clearly an ID, not part of the API. Because it follows **subscription_items**, it will be a subscription item ID. So the path needs to include that ID. Use this as your path:

```
/subscription_items/{subscription_item}/usage_records:
```

3. Next, notice that above the curl sample, it says that the method is POST, so add the post method.
4. Copy the description from the documentation. Start with a "pipe" to indicate that you will use multiple paragraphs.

   ```
   description: |
   ```

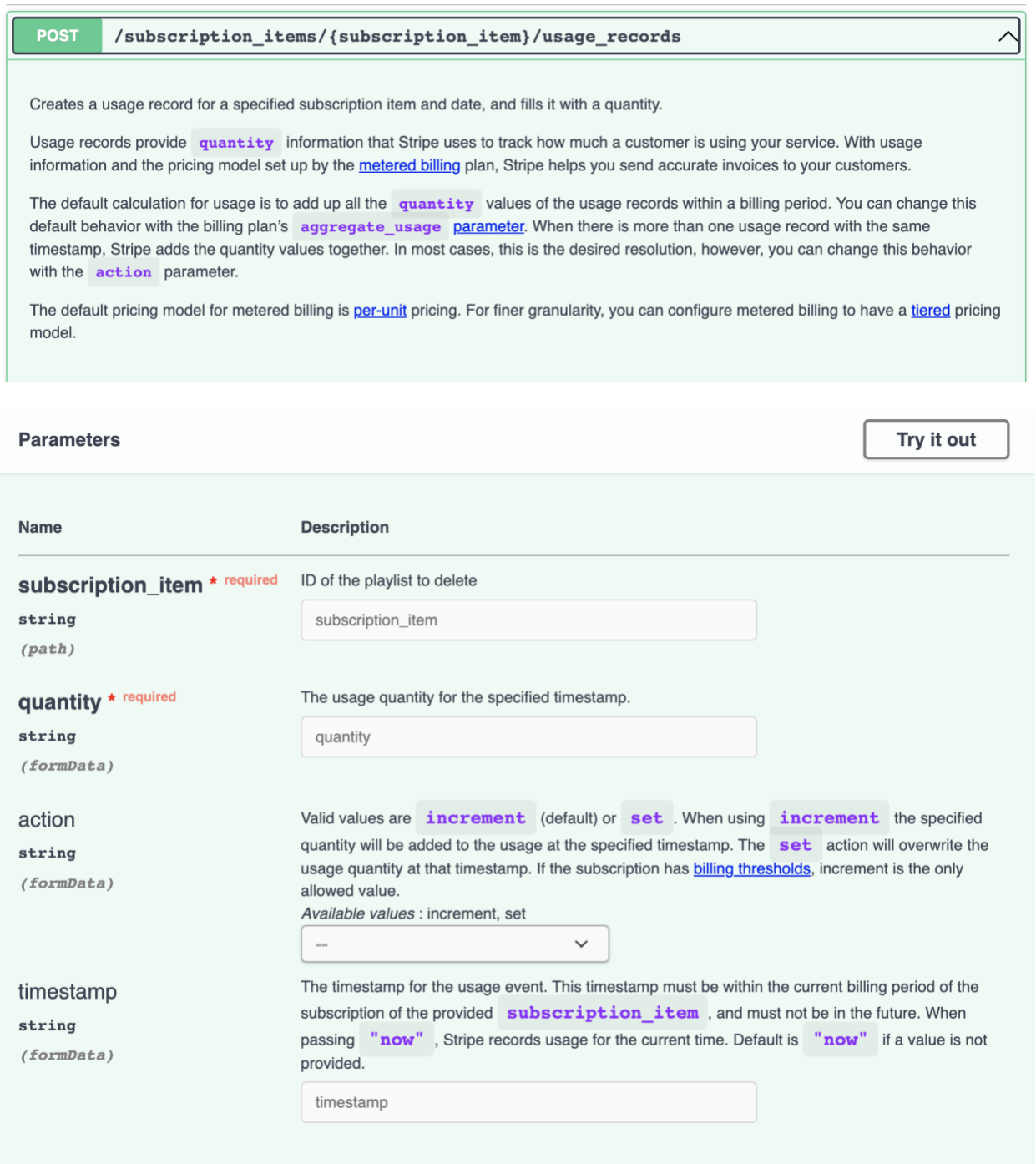
   Use the techniques you've used before to add links and monospace font. Put blank lines in between paragraphs. (Note that the editor will get confused by the # signs in the URLs and think the they are comments, making them gray. But if you look at the right side, you will see it displayed correctly.)

5. Now add a **parameters** key. The curl command has -d with the format x=y, which is called formData. This is like query parameters, but in the POST body.
6. Add the first parameter, which is the ID. It will have a **name** of **subscription_item**, an **in** of **path**, a **required** of **true**, a **type** of **string**, and a **description** of **The ID of the subscription item for this usage record.**
7. The second parameter is **quantity**. It will have an **in** value of **formData**, and a type of string. Take the description from the documentation. Set it as required.
8. Now add similar parameters for **action** and **timestamp**. Note that these are optional.
9. Note that **action** is an enumeration with specific values. Add this to the **action** section to specify that:

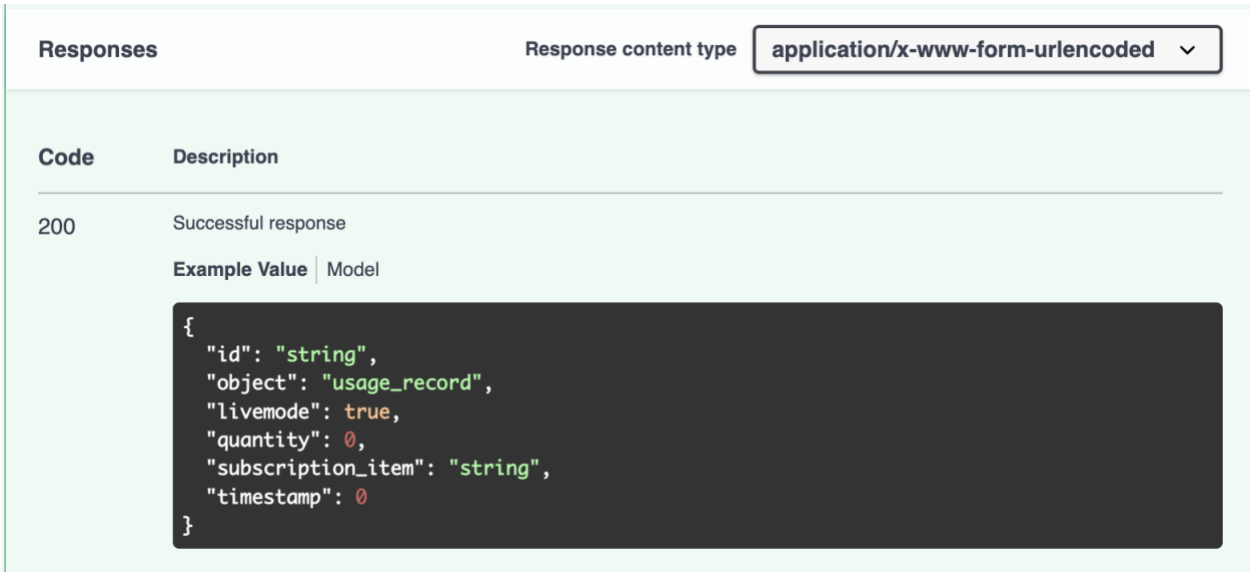
   ```
   enum:
     - increment
     - set
   ```

10. Finally, add the response by adding a **responses** section under the **parameters** section of the **post** section. Then add a key of **200** and a description of **Successful response**. It returns a usage record object, so add a **schema** key and a **$ref** that points to that object.

At this point, your errors should be totally gone. If you have errors, read them and see if you can figure them out. Often it's indentation problems, or something in the wrong section. This is what your documentation should look like for the POST:



If you click on **Try it out** and then select the dropdown box in the **action** section, you'll see that you can choose either **increment** or **set**.



Note the red asterisks indicating that a parameter is required. At the top, you will find an authorization section.

Click on **Authorize**, and it will bring up a dialog to enter basic authentication credentials (i.e., username and password).

If you need help, you can look at my solution so far at: http://sdkbridge.com/swagger/stripe3.yaml

List all subscription item period summaries

Scroll down again, and you will see the second request, which is **List all subscription item period summaries**. To understand what some of the parameters mean, read the section on Pagination: https://stripe.com/docs/api/pagination?lang=curl

Follow these steps to create the second request:

1. Copy everything from your first path (**/subscription_items/{subscription_item}/usage_records**) and paste it below that section to create the second path.
2. To get the path name, look at the URL of curl sample request. It has the same host and base URL, so we just need everything between that and the question mark. Rename the path to be **/subscription_items/{subscription_item}/usage_record_summaries**
3. If you look at the curl sample, you will -G, which means GET, so change **post** to **get**.
4. Change the description to match the documentation.
5. Look under **Parameters** in the documentation to find the parameters. Leave the description of the **subscription_item** parameter the same, since it is identical to the first API request.

6. Add an **ending_before** parameter with an **in** of **query**, a **required** of **false** (it's optional), and a **type** of **string**. Why string? Well, this is kind of a guess. It should really be specified, but if you read the documentation, it's an ID, and Stripe appears to use strings for IDs. Finally, copy the description from the documentation.
7. Add a **limit** parameter. You should be able to figure out its **in**, **required**, and **type**. In addition, you can add a **maximum** and **minimum**.
8. Copy the **ending_before** parameter and add it to the end of the parameters list. Change its name to **starting_after** and update its description. (Everything else is the same.)
9. For the response, you will need to create a new object in the **definitions** section. Let's call it **usageRecordSummary**. First change the schema to

```
$ref: '#/definitions/usageRecordSummary'
```

10. Now create it in the **definitions** section. For the **description**, use the documentation.
11. The best way to figure out what goes into this object is to look at the curl example. There are four properties: **object**, **url**, **has_more**, and **data**.
12. **object** is a **string** with a value of "list". You can copy the object from **usageRecord**, but change the **enum** value.
13. **url** is a **string** that refers to the URL of these summaries. So have a **description** of **URL of the subscription item summaries.**
14. **has_more** is a **boolean** that is described here: https://stripe.com/docs/api/pagination?lang=curl#pagination-has_more
15. **data** should refer to an array of new objects. You can tell it's an array because of the square brackets in the sample. This means that it will be of type **array**, and it will refer to a new object, which you can add to **definitions**. You can call this new object **usageRecordData**.
16. From here, see if you can figure out how to create the **usageRecordData** object by looking at the curl sample. The types are either strings or integers. (Except **period**, which will be of type **object**.) The **start** value is set to **null** in the example (meaning, no value at all), but assume that one is an integer.

Note: It's not clear from the documentation what to put in all of the descriptions. Here is my best guess based on reading various parts of the documentation.

Key	Description
id	ID of the usage record.
object	String representing the object's type. Objects of the same type share the same value.
invoice	ID of the invoice.
livemode	Has the value `true` if the object exists in live mode or the value `false` if the object exists in test mode.
period	Time period.
end	Ending timestamp.
start	Starting timestamp.
subscription_item	ID of the subscription item.
total_usage	The total number of usage records in the billing plan.

Here is my final solution: http://sdkbridge.com/swagger/stripe4.yaml