A brief UberRUSH overview and tutorial

Justus Perlwitz

2016-06-15

Contents

Uber Rush provides cost-effective on-demand courier services. It is an exciting service that will allow companies to start delivering to local customers faster. Uber tries to make the process as easy as possible by providing an API which can be easily integrated into existing shop solutions and lets customers order cheap and easy shipping. For a high-level overview, please refer to this document.

We are going to look at the UberRUSH API from a technical perspective and try coming up with common usage scenarios that an application developer will encounter when working with this API. This article is going to be useful if you plan on integrating UberRUSH into your application. Instead of working with the real API, we will make use of Uber's API sandbox.

We want to perform the following steps:

1. Log in to the Uber API. We want to retrieve a valid OAuth2 token.
2. Request delivery quotes. Customers need to see a delivery quote before choosing UberRUSH delivery.
3. Schedule a delivery. In order to start the delivery process, the delivery endpoint will be called.
4. Retrieve delivery details. Present the customer with shipping information and forward the shipping info to shipping/warehouse.
5. Cancel the delivery. Our customer regrets their purchase. We are in a sandbox. Let's complete the cycle and cancel the delivery.

1 Signing up for Uber's API

No API that is worth its two cents can be used without a good old API key pair. First we head over to the UberRUSH account management page. Then, we are ready to register our application and create a key pair. Head over to the Developer Dashboard and you'll be done in no time.

2 API Wrapping with Tortilla + Python

Now, I would like to introduce a tasty, delicious Python-based API tool called Tortilla. It will make any inedible API digestible (as long as it does json). Similar to how SQLAlchemy or the Ruby on Rails ORM are designed you can compose queries in a compact way. For example, given a Tortilla wrapped API `taco`, POSTing a JSON object to an endpoint called

`/user/1/account`

is as easy as doing the following

```
taco.user(1).account.post(data=dict()).
```

3 Access Token retrieval

For authentication, we are going to follow the OAuth2 guide in UberRUSH's API documentation.

Before getting started, let me vent about the quality of the API. Uber requires developers to always use OAuth2 tokens. Entirely reasonable for a professional application, you might say. What is not reasonable, is this:

4 How to make the Uber API crash and burn

It should not be possible to make an API endpoint return `500`. But it is. Take a look for yourself:

```
curl 'https://login.uber.com/oauth2/v2/token' \
    -H 'Content-Type: application/json' \
    -d '1' -v 2>/dev/null
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>500 Internal Server Error</title>
<h1>Internal Server Error</h1>
<p>The server encountered an internal error and was unable to complete your
request. Either the server is overloaded or there is an error in the
application.</p>
```

Woops! It was an accident, I swear!

5 How to actually log in

At this point I am tired of trying to figure out why the Uber API works with curl, but not with form-encoded requests in Tortilla. Human patience only gets you so far. That's why I am calling curl as a subprocess here. The response will be stored as a dictionary in `token`. Furthermore, I am going to assume that the API key pair is stored in the environment.

```
from os import getenv
from subprocess import check_output
from json import loads
token = loads(check_output((
    'curl',
    '-F', 'client_secret=' + getenv('RUSH_CLIENT_SECRET'),
    '-F', 'client_id=' + getenv('RUSH_CLIENT_ID'),
    '-F', 'grant_type=client_credentials',
    '-F', 'scope=delivery',
    'https://login.uber.com/oauth/v2/token')).decode())
print(token.keys())
# TODO understand why any sane API provider would want a
# 1 month expiration time 2016-06-14
print("Token expires in {} days".format(token['expires_in'] / 3600 / 24))
```

```
dict_keys(['access_token', 'scope', 'last_authenticated', 'token_type',
           'expires_in'])
Token expires in 30.0 days
```

That means we have completed **step 1**.

6 Retrieving a delivery quote

Now that we have a valid token, let's wrap the Uber API sandbox with Tortilla and get started making some API calls. Tortilla takes a URL endpoint and an arbitrary amount of headers. While it is not well documented, I appreciate the fact that headers can be preset for all requests to follow. In this instance we set up an `Authorization` header using the access token we have retrieved in the last step.

```
from tortilla import wrap
uber = wrap("https://sandbox-api.uber.com/v1",
            headers={'Authorization': 'Bearer {}'
                     .format(token['access_token'])})
```

Tortilla does some mighty crazy `__getattr__` magic in the background. You will now see how this can be useful.

Let's pretend for a second that Google Guy™ has just ordered the new iPhone 8 and selected UberRUSH for delivery. In order to present a shipping quote, we need both the pickup and drop off address. Pickup will be known to us since we are the vendor. Obviously, it is going to be shipped right from our Apple headquarters to the Googleplex in Mountainview, California. I've taken the liberty of jotting down both addresses for you:

```
pickup = {'location': {'address': '1 Infinite Loop',
                       'city': 'Cupertino',
                       'state': 'CA',
                       'postal_code': '95014',
                       'country': 'US'}}
dropoff = {'location': {'address': '1600 Amphitheatre Parkway',
                        'city': 'Mountainview',
                        'state': 'CA',
                        'postal_code': '94043',
                        'country': 'US'}}
```

According to the API documentation, we need to POST the two addresses to the following endpoint:

```
/v1/deliveries/quote
```

Now we can inform our Google employee on what shipping with UberRUSH is going to cost.

```
response = uber.deliveries.quote.post(data={'pickup': pickup,
                                            'dropoff': dropoff})
print("The delivery fee will be: {} USD".format(response.quotes[0].fee))
```

```
The delivery fee will be: 5.0 USD
```

For completeness' sake, here is the full response.

```
{'quotes': [{'currency_code': 'USD',
   'dropoff_eta': 13,
   'end_time': None,
   'estimated_at': 1465973935,
   'expires_at': 1465975735,
   'fee': 5.0,
   'pickup_eta': 8,
   'quote_id': 'db9434b3-51b7-4a91-ae4b-6d19a4d41ccc',
   'ready_by_time': None,
```

```
   'start_time': None},
  {…},
  {…},
  {…}]}
```

Uber is giving us four different quotes, all for 5 USD but with different pick up times. I have taken the liberty of abbreviating the three last quotes.

Let's choose the first `quote_id` in the list and schedule it. We will find out how to do it in the next section.

```
quote_id = response.quotes[0].quote_id
print('quote_id: {}'.format(quote_id))
```

```
quote_id: db9434b3-51b7-4a91-ae4b-6d19a4d41ccc
```

For now, we have completed **step 2**.

7 Scheduling a Delivery

We need to specify the item to be delivered, using the `/deliveries/` endpoint. In this case we will tell them that we're shipping an iPhone, as we promised to deliver to Google Guy™ .

```
items = [{'title': 'iPhone 8',
          'quantity': 1,
          'width': 2.64,
          'length': 5.44,
          'height': 0.27,
          'price': 549.0,
          'currency_code': 'USD'}]
```

8 Seriously, Uber?

Those Uber barbarians use inches in their POST parameter docs. The banality of API design.

Furthermore, letting me specify the price as a float is a crude mistake. No way you should trust IEEE floating point decimals to your billing logic. This sounds like something right out of Coding Horror. Actually, there is something on Coding Horror about this.

Less rambling. More API calling.

```
delivery_pickup = {
    'contact': {'first_name': 'Tim',
                'last_name': 'Doe',
                'email': 'tim.doe@apple.com',
                'phone': {'number': '+1 (408) 996-1010',
                          'sms_enabled': False}},
                **pickup}  # Python 3.5 dict unpacking, see PEP 448
delivery_dropoff = {
    'contact': {'first_name': 'Larry',
                'last_name': 'Doe',
                'email': 'larry.doe@google.com',
                'phone': {'number': '+1 (650) 253-0000',
                          'sms_enabled': False}},
                **dropoff}
response = uber.deliveries.post(
    data={'quote_id': quote_id,
          'items': items,
          'pickup': delivery_pickup,
          'dropoff': delivery_dropoff})
```

Our delivery was successfully scheduled and we receive the following answer.

```
{'courier': None,
 'created_at': 1465973936,
 'currency_code': 'USD',
 'delivery_id': 'bd108436-404c-4264-95fe-e46bdebd27be',
 'dropoff': {…},
 'fee': 5.0,
 'items': [{'currency_code': 'USD',
   'height': 0.27,
   'is_fragile': False,
   'length': 5.44,
   'price': 549.0,
   'quantity': 1,
   'title': 'iPhone 8',
   'weight': 0.0,
   'width': 2.64}],
 'order_reference_id': '',
 'pickup': {…},
 'quote_id': 'db9434b3-51b7-4a91-ae4b-6d19a4d41ccc',
 'status': 'processing',
 'tracking_url': 'https://api.uber.com/v1/sandbox/map'}
```

Again, I've shortened a few attributes as we've just specified these above. We have succesfully scheduled a delivery. **Step 3** is complete.

9 What did we just schedule?

Is a UberRUSH courier going to come to Apple's offices now? I think I'm going to be in trouble. Let's double check and see whether my request has been entered into Uber's system. We will use the `/deliveries` endpoint .

```
response = uber.deliveries.get()
response.deliveries[0]
```

```
{'courier': None,
 'created_at': 1465973936,
 'currency_code': 'USD',
 'delivery_id': 'bd108436-404c-4264-95fe-e46bdebd27be',
 'dropoff': {…},
 'fee': 5.0,
 'items': [{'currency_code': 'USD',
   'height': 0.27,
   'is_fragile': False,
   'length': 5.44,
   'price': 549.0,
   'quantity': 1,
   'title': 'iPhone 8',
   'weight': 0.0,
   'width': 2.64}],
 'order_reference_id': '',
 'pickup': {…},
 'quote_id': 'db9434b3-51b7-4a91-ae4b-6d19a4d41ccc',
 'status': 'processing',
 'tracking_url': 'https://api.uber.com/v1/sandbox/map'}
```

Again, abbreviations were made for the sake of brevity.

10 Delivery detail view

Looks good. Let's get a more detailed view on this delivery using the `/deliveries/{delivery_id}` endpoint

```
delivery_id = response.deliveries[0].delivery_id
print('delivery_id: {}'.format(delivery_id))
response = uber.deliveries(delivery_id).get()
print('Delivery  status: {}'.format(response.status))
```

```
delivery_id: bd108436-404c-4264-95fe-e46bdebd27be
Delivery  status: processing
```

The order has a status of `processing`. According to the API docs, this means that a courier is about to be dispatched. **Step 4** is complete.

11 Cancelling deliveries

No time to waste; we shall cancel this sandbox delivery using the `deliveries/{delivery_id}/cancel` endpoint.

Since we are thorough beings, we are going to make sure all deliveries are cancelled, just in case any previous deliveries are left. You will see more than one delivery here as this is not the first session I am spending with the UberRUSH API.

```
from operator import itemgetter

deliveries = uber.deliveries.get().deliveries
delivery_ids = map(itemgetter('delivery_id'), deliveries)
for delivery_id in delivery_ids:
    print('Cancelling delivery_id: {}'.format(delivery_id))
    response = uber.deliveries(delivery_id).cancel.post()
```

```
Cancelling delivery_id: bd108436-404c-4264-95fe-e46bdebd27be
Cancelling delivery_id: bd50916a-f16c-4dc8-819d-aa1cfc0afa8d
Cancelling delivery_id: d8b1a609-be94-43f8-ae55-fd6d37975771
Cancelling delivery_id: 93eea5aa-b27d-419f-8552-8d43b9050d25
Cancelling delivery_id: b56be06c-5bd1-4ac2-932c-75f97d90241e
Cancelling delivery_id: 59952629-edad-4c71-827d-523345a1ca6f
Cancelling delivery_id: 0ee33f16-c9da-42e6-95d6-6a5fbc1c4ebe
Cancelling delivery_id: 7c87af43-433b-407e-b6a3-2077b58020b4
Cancelling delivery_id: cde657ba-503c-43d8-851c-273a41f39bf3
Cancelling delivery_id: 83c01bb8-eb8e-455f-b0b5-a310b9e9767e
Cancelling delivery_id: 62046167-a51c-471c-a67a-1495bb6ee0a6
Cancelling delivery_id: 1cf24365-8f9a-46db-95b0-87e2018454ad
Cancelling delivery_id: faf3ebbf-7934-45df-abc1-1598d7acdaac
Cancelling delivery_id: 9fd3bfa4-83b2-4b2a-a7b3-de10c6532bf7
Cancelling delivery_id: e052d01e-bea3-4a2d-b890-0ce38c24c519
Cancelling delivery_id: fd223605-e0ef-418b-8662-774d9ec316b7
Cancelling delivery_id: 271866db-d2d3-40b1-a274-f4e69682cd62
Cancelling delivery_id: a2a2b3d4-c39e-4d58-9d03-9e48a738fd8b
Cancelling delivery_id: b6f67b89-8982-49a3-87c6-a90778bae203
Cancelling delivery_id: 93f6b16d-2338-4866-8e59-ba6a318cdc27
Cancelling delivery_id: 206950cf-0580-4079-b8de-28e3d623dd4a
Cancelling delivery_id: 6c41bfd5-610c-4a57-9fc3-b531a649ad02
Cancelling delivery_id: 7d1c219e-1f34-41ee-89bf-30055dff4873
Cancelling delivery_id: 30f345ee-3b2e-47a0-88e1-9ced2a5e68fd
Cancelling delivery_id: 93195b11-973c-418d-8e1e-247555e4b91f
Cancelling delivery_id: af1a4e0e-34db-4919-940e-4f0d147ae941
```

This is where it gets a little interesting. Instead of deleting the delivery, UberRUSH will just mark it as `client_canceled`. We can verify this using the `/deliveries/{delivery_id}` endpoint once again:

```
print(uber.deliveries(delivery_id).get().status)
```

```
client_canceled
```

Good. Google will not get their iPhone 8 anymore and **step 5** is complete.
We've now gone through the whole UberRUSH cycle in one sitting! What's the verdict?

12 Summary

It is exciting to think about the potential uses for this easy-to-use API. Ordering flowers and having them instantly shipped to your apartment while cutting out all the messy delivery middleware? Having hipster bike couriers bring artisanal craft beer? All this is going to be possible with UberRUSH! Farewell, Spoonrocket!