Program :	***Platform Integration***
Project :	***Plateforme Taxi Montréal***
Deliverable:	***Le.Taxi documentation by VDM/BTM***

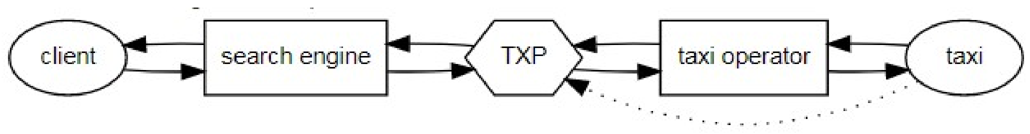
History

Version	Description	Author	Date
1.0	Original draft	David Beaudoin	12/12/2017

1. Introduction

1.1 Overview

The **Taxi Exchange Point** (**TXP**) aim is to connect taxis and their clients. Clients can use taxis **search engines** to hail taxis geolocated by **taxis operators**. The Montreal Taxi Platform mediates between search engines and operators.



The solution offered by the BTM is based on the solution used in France (http://le.taxi/). The new solution kept full compatibility with the original solution to offer possibilities for search engine to expand to other city in the future.

1.2 Process of integration

The search engine will contact the TXP and receive an API key for the acceptation environment. All the tests will be made with a fake operator associated with the search engine so that no one interfere in each other tests. Also, the search engine must tests all of the scenarios presented in section 4.3 Hail Scenarios and give access to the application in acceptation to the BTM to verify conformity of the application. Once the tests are positive, the search engine will be given an API key for production.

The TXP administrator can be contacted at: support.taxi.exchange.point@ville.montreal.qc.ca

Here are the links to communicate with the TXP services:
Acceptation : https://taximtl.accept.ville.montreal.qc.ca
Production : https://taximtl.ville.montreal.qc.ca

1.3 Mandatory HTTP Headers

The following HTTP Headers are mandatory for all requests to the TXP REST APIs:

Name	Value	Description
`Accept`	`application/json`	*media types which are acceptable for the response*
`X-VERSION`	`2`	*Version of the API*
`X-API-KEY`	token	*API Key*

1.4 Authentication

Authentication of your application is done for each query to the TXP by including a HTTP header `X-API-KEY`.
API keys are available for accredited developers and will be distributed by the BTM (Bureau Taxi Montréal) upon demand and validation.

1.5 APIs implementation

This documentation provides an overview of the TXP REST APIs. REST APIs provide access to resources (data entities) via URL paths. To use a REST API, your application will make an HTTPS request and parse the response. Your methods will be the standard HTTP methods like `GET`, `PUT` and `POST`. REST APIs operate over HTTPS making it easy to use with any programming language or framework. The input and output format for the TXP REST APIs is JSON.

2. Roles

2.1 Search Engines

There are two main interactions between **Taxis Search Engines** and the **Taxi Exchange Point (TXP)**: looking for taxis around a client, and hailing a taxi for a client.
Warning: in accordance with the Terms of Service, you are allowed to look for a taxi or hail one *only* within the context of a real-time interactive demand of a real customer: automation, replay and any other form of mass data gathering is expressly forbidden. Also note that while all clients are anonymous to the TXP, the search engine has the responsibility to prevent false demands (ex.: ban malicious users).

2.1.1 Looking for taxis around a client

In order to obtain the list of all taxis available around a client, your application does an HTTPS Get request on the taxis API.
That API sends back all the available taxis around the `{lon,lat}` location of the client with all their characteristics. Only the taxis that are can be hailed are returned.

```
GET /api/taxis

Example: /api/taxis?lat=45.511885&lon=-73.607919

Parameters

query

lon                     (string) (required)

lat                     (string) (required)

favorite_operator       (string) (not used in Quebec)

count                   (integer) (set the number of taxi to return. Default 10)

Response

{

    "data": [

        {

            "ads": {

                "insee": "102011",

                "numero": "10846_1512414871"

            },

            "crowfly_distance": 0.00145,

            "driver": {
```

```
            "departement": "660",
            "professional_licence": "3052_1512414871"
```

```
        },
        "id": "2KWGs96",
        "last_update": null,
        "operator": “,
        "position": {
            "lat": null,
            "lon": null
        },
        "private": false,
        "rating": 4.5,
        "status": "free",
        "vehicle": {
            "characteristics": [
                        "luxury",
                        "credit_card_accepted",
                        "nfc_cc_accepted",
                        "amex_accepted",
                        "bank_check_accepted",
                        "fresh_drink",
                        "dvd_player",
                        "tablet",
                        "wifi",
                        "baby_seat",
                        "bike_accepted",
                        "pet_accepted",
                        "air_con",
                        "electronic_toll",
                        "gps",
                        "every_destination",
                        "special_need_vehicle"
              ],
            "color": "black",
            "constructor": "Toyota",
            "licence_plate": "7254_1512414872",
                "model": "accepted_by_customer_after_timeout",
                "nb_seats": 3
            }
        },
…]}
```

Key	**Value Type**	**Description**
`vehicle`	vehicle	*A partial* vehicle *object with only the fields:* `characteristics (see the list of characteristics following this table)`, `color`, `constructor`, `licence_plate`, `model`, `nb_seats (excluding driver)`. *Warning: some of those fields might not be returned (or be returned with a* `null` *value) if they were not provided by the taxi operator.*
`ads`	ADS	*A partial* ADS *object with only the fields:* `insee`, `numero`.
`driver`	driver	*A partial* driver *object with only the fields:* `departement`, `professionnal_licence`.
`id`	string	*A long-lived identifier generated for this* `operator`/`vehicle`/`ads`/`driver` *tuple by the TXP.*
`crowfly_distance`	float	*The crow fly distance between the taxi and the customer. (km)*
`operator`	string	*The name of the operator.*
`private`	boolean	*As per VDM and BTM's requirements, as an option, operators can set the taxi's private field to true or false. By default, the taxi's private field is set to false.* A private taxi will never receive hails from the TXP. Search engine will only see those with private value set to false.
`rating`	float	*The mean of the ratings of last rides of the taxi.* *It is calculated by the TXP and falls between 0 and 5.*
`status`	status	*Status of the taxi.* *The possible values are described in the section 4.1 Hail Status.* Hail_Status
`position`	{lat, lon}	*The latitude and longitude of the taxi.*
`last_update`	integer	*Timestamp of the last geolocation update of the taxi. The format is the usual Unix time (IEEE P1003.1 POSIX) and as such is UTC (no timezone).*

List of vehicle characteristics

air_con	*This vehicle is equipped with air conditioning.*
amex_accepted	*This vehicle accepts American Express card for any amount (no minimum).*
baby_seat	*This vehicle is equipped with a baby seat.*
bank_check_accepted	*This vehicle accepts national bank checks (foreign bank checks might still be refused).*
bike_accepted	*This vehicle can transport a bicycle.*
credit_card_accepted	*This vehicle accepts credit card payments for any amount (no minimum).* *This should be true for vehicle accepting at least Visa and MasterCard. There is a different Boolean amex_accepted for American Express.*
dvd_player	*This vehicle has a DVD player at the disposal of clients during the ride.*
electronic_toll	*This vehicle is equipped with an electronic device letting them use express toll booths on toll roads.*
every_destination	*As per the French regulation, taxis can refuse service to clients whose destination is not within their zone. Some taxis do accept any destination outside of their zone. The every_destination boolean should be false by default, and true for taxis who renounce their right to refuse service to clients depending on their destination.*
fresh_drink	*This taxi offers refreshments.*
gps	*This vehicle is equipped with GPS navigation.*
luxury	*This is a luxury vehicle.*
nfc_cc_accepted	*This vehicle accepts NFC credit card*

	payments.
pet_accepted	*This vehicle can accommodate pets (understood as cats or small dogs; other large or unusual pets might still be refused).*
special_need_vehicle	*Wheelchair accessible vehicle as defined in “EU/678/2011” (which amends 2007/46/EC).* *Vehicle constructed or converted specifically so that they accommodate one or more persons seated in their wheelchairs when travelling on the road.*
tablet	*This vehicle has a digital tablet at the disposal of the clients during the ride.*
wifi	*This vehicle has complimentary Wi-Fi aboard.*

2.1.2 Hailing a taxi

Once you have received the list of all taxis available around a client, you can hail one of those taxis. That Hail is created by doing a HTTPS POST request to the hails API. The JSON object `hail` to be created has to be in the data section of the request. In case of success, the response contains the newly created JSON object `hail` including its newly created *id.* You must store this id in order to query and update the status of the hail.

```
POST /api/hails/
Parameters
body (JSON)
{
  "data": [
    {
      "customer_lat": 45.58017,
      "customer_lon": -73.61479,
      "customer_address": "801 rue Brennan, Montreal QC H3C 0G4",
      "taxi_id": "taxi1111",
      "operateur": "coop",
      "customer_phone_number": "514 555-6565",
      "customer_id": "anonymous"
    }
  ]
}
```

Response (JSON) Status 200 OK

```
{
  "data": [
    {
      "creation_datetime": "Thu, 22 Dec 2016 11:24:53 -0000",
      "customer_address": "801 rue Brennan, Montreal QC H3C 0G4",
      "customer_id": "anonymous",
      "customer_lat": 45.58017,
      "customer_lon": -73.61479,
      "customer_phone_number": "514 555-6565",
      "hail": [
        {
          "rating": 4.5
        }
      ],
      "id": "hail1111",
      "incident_customer_reason": null,
      "incident_taxi_reason": null,
      "last_status_change": "Thu, 22 Dec 2016 11:24:53 -0000",
      "operateur": "coop",
      "rating_ride": null,
      "rating_ride_reason": null,
      "reporting_customer": null,
      "reporting_customer_reason": null,
      "status": "received",
      "taxi": {
        "id": "taxi1111",
        "last_update": 1482423893,
        "position": {
          "lat": 45.6164341134,
          "lon": -73.6138161294
        }
      },
      "taxi_phone_number": null,
      "taxi_relation": {
        "rating": 4.5
      }
    }
  ]
}
```

Key	Value Type	Description
`customer_lat`	float	*Latitude of the position of the client.* *This should be in JavaScript double precision floating-point format, with decimal separator "."*
`customer_lon`	float	*Longitude of the position of the client.* *This should be in JavaScript double precision floating-point format, with decimal separator "."*
`customer_address`	string	*Address of the position of the client.* *This address will be used by the taxi driver to find the client.* *It should be displayed and validated by the client.(mandatory)* *Warning: In some cases, a place description might be more meaningful than a postal address.*
`customer_id`	string	*Identifier of the customer.* *This identifier is an identifier the Search Engine uses to uniquely identify the client. It can be a database id, an IMEI, a cookie identifier, a hash of a phone number, or any long lived identifier which will stay the same when the same client*

		comes back. *It does not need to carry any signification for the TXP: the TXP does not need the identity of the client.* The only acceptable value is “anonymous” (case sensitive)
`customer_phone_number`	string	*Phone number of the client. (Mandatory)* *This phone number might be used by the Operator of the taxi in case it proves difficult to find the client.*
`taxi_id`	string	*Identifier of the taxi the client is hailing.* *This identifier was returned by the TXP in the* Taxi *object.* *Warning: for historical reasons, when a “**search engine**” sends a new hail to the TXP, the taxi id should be passed as a “`taxi_id`” field directly in the “`hail`” object.* *In all subsequent exchanges, including when the “TXP” forwards the “`hail`” to the “**operator”**, the taxi id appears instead as an “`id`” field in an embedded “`taxi`” JSON object inside the `hail`.*
`operateur`	string	*Identifier of the Operator of the taxi the client is hailing.* *This identifier was returned by the “TXP” in the “*Taxi*” object.*
`status`	status	*Status of the hail.* *All possible values are described here*
`id`	string	*Identifier of the hail.* *This identifier should be `null` or omitted when a “**search engine**” sends a new hail to the TXP. The newly generated “`id`” will be in the*

		"`hail`" object returned by the TXP as a response.
`taxi`	taxi	*Details of the taxi selected for the hail.*
`taxi_phone_number`	string	*Phone number the client should call in case of problem.* *This phone number can be either the number of the call center of the operator or the mobile phone number of the taxi driver. It has to be reachable at the time of the ride: call center numbers should only be transmitted during opening times.*
`incident_customer_reason`	string	*Reason for the incident that prompted the client to cancel the ride.* *This is reserved for future use. The only accepted value as per version 2 of the API is an empty string. This field should be used by "**search engines**" when setting the status of the ride to `incident_customer`.*
`incident_taxi_reason`	string	*Reason of the incident that prompted the taxi to cancel the ride.* *This field should be used by "**operators**" when setting the status of the ride to `incident_taxi`.* *The possible reasons are: "`no_show`" (when the client cannot be found), "`address`" (when the address cannot be found), "`traffic`" (when a traffic jam prevents the taxi to arrive at the location in a reasonable time) and "`breakdown`" (in case of a mechanical problem on the vehicle preventing the taxi to continue operating). This information will be visible too the search engine querying the `hail`.*
`rating_ride`	Integer (from 1 to 5)	*Rating of the ride by the client.* *This field should be used by "**search engines**" during or after the ride to let customers rate the ride between 1*

		and 5 stars. A ride can be rated multiple times, but only the latest rating will be considered.
`rating_ride_reason`	string	*Explanation of the rating of the ride by the client.* *This field should be used by "**search engines**" during or after the ride to let the client explain low ratings of the ride. It is recommended to ask customers for their "`rating_ride_reason`" when their `rating_ride` is 3 stars or less. This information will not be individually transmitted to the taxi driver.* *The possible values are "`ko`" (the taxi never showed and/or the ride did not happen), `payment` (credit card refused, etc), `courtesy,` (general attitude problems, loud radio, etc), `route` (subpar itinerary, etc) and `cleanliness` (dirty car, cigarette smell, etc).*
`reporting_customer`	boolean	*Reporting of a problem encountered with a customer by a driver.* *This field should be used by "**operators**" during or after the ride to let the taxi driver inform the search engine that a problem happened with the client. In that case, the hail should be updated with a `reporting_customer` set to `True` and a `reporting_customer_reason` should be provided.*
`reporting_customer_reason`	string	*Explanation of the problem encountered with a customer by a driver.* *This field should be used by "**operators**" during or after the ride to let taxi drivers explain the type of problem they encountered with a client. It is only required if "`reporting_customer`" is set to `True`.* *The possible values are "`ko`" (the client was nowhere to be seen and/or the ride did not happen),*

		“payment” (unpaid ride, bargaining, etc), courtesy (general attitude problems, etc), route (non existing destination address, etc) and cleanliness (dirty luggages, cigarettes, etc).

2.1.3 Querying the status of a Hail

In order to keep track of the status of the hail, you can do a HTTPS GET request to the hails API. The identifier of the hail is the *id* that was returned in the response to the POST request used to create the hail (see Hailing a taxi).

The search engine needs to fetch the status of the hail to check if the status changed. The polling delay for this check must be equal or superior to 5 seconds when the status is between “emitted” and “accepted_by_taxi”. The polling delay should be equal or superior to 120 seconds when the status of the hail reachs the “accepted_by_taxi” status until it reach “customer_on_board”.
The list of all possible statuses is available below.

The JSON object has the same structure as the response described in section 2.1.2.

```
GET /api/hails/{hail_id}

Parameters

Path

Hail_id (string)(required)
```

2.1.4 Updating a Hail

In order to finalize the transaction, the client has to reconfirm the hail after it is acknowledged by the taxi. It is done through a call of the HTTPS PUT request to the hails API.
The status of the hail has to be `accepted_by_taxi` before the client is asked to reconfirm. If you don't send a PUT request with either the `accepted_by_customer` or the `declined_by_customer` status in the next 30 minutes, the hail will automatically be updated to `timeout_customer` and the whole transaction be considered cancelled. You do have to check when you do a PUT with the status `accepted_by_customer` that the status in the response is indeed `accepted_by_customer` (you reconfirmed and the taxi is arriving) and not `timeout_customer` (your reconfirmation arrived too late, the whole transaction had to be canceled).

The JSON object has the same structure as the response described in section 2.1.2.

```
PUT /api/hails/{hail_id}

Parameters
```

Path

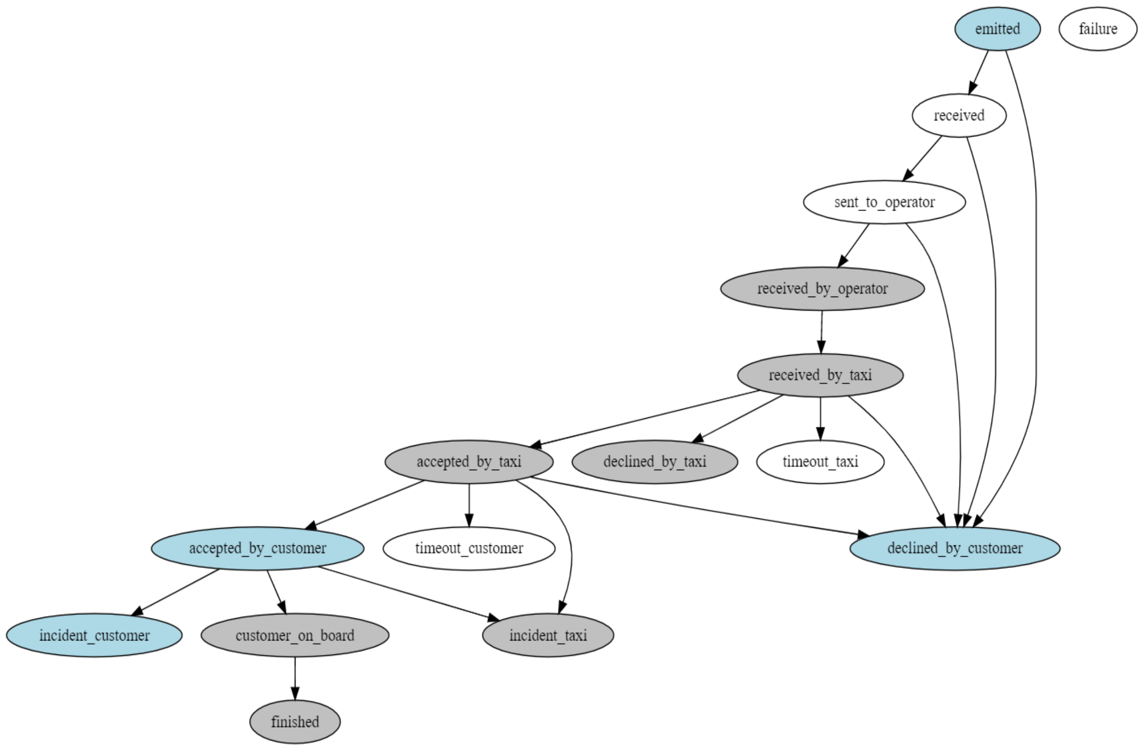
```
Haild_id (string)(required)
```

Body (JSON) ***all value inside the JSON are OPTIONAL, use as needed*

```
{
  "data": [
    {
      "status": "accepted_by_customer",
      "incident_customer_reason": "",
      "customer_lat": 0,
      "rating_ride": 0,
      "rating_ride_reason": "ko",
      "customer_lon": 0,
      "customer_address": "string",
      "customer_phone_number": "string",
    }
  ]
}
```

3. Hails

3.1 Hail Status



The states in gray can be reached after an interaction with the operator. The ones in blue can be reached after an interaction with the search engine. The states in white are under the control of the TXP.

The section 4.3 "Test Scenario" describes the scenario that the search engine must support in order to receive a hail from the TXP.

Value	Description	Interaction
`emitted`	*The initial status of hail when created by a search engine*	*This is the status that should be used in the payload of the `POST` request on the /hails/ API when a **search engine** creates a new hail.*
`received`	*The hail is received from the **search***	*The TXP changes the status to `received` after having sent back the*

	***engine** by the TXP*	*complete hail (with the newly generated id) to the **search engine** and before forwarding the hail to the **operator**. Time before failure 15 seconds.*
`sent_to_operator`	*The hail has been sent from the TXP to the operator*	*The TXP changes the status to `sent_to_operator` after forwarding the hail to the **operator** endpoint. Time before failure 10 seconds.*
`received_by_operator`	*The **operator** has acknowledged receiving the hail from the TXP*	*The TXP changes the status to `received_by_operator` after receiving an HTTP 200 response from the "**operator**" endpoint. Time before failure 10 seconds.*
`received_by_taxi`	*The hail has been received by the taxi*	*The "**operator**" should set the status of the hail to `received_by_taxi` by doing a "`PUT`" request on the /hails/{hail_id} API when the hail has been presented to the taxi driver. Time before timeout_by_taxi 30 seconds.*
`accepted_by_taxi`	*The hail has been accepted by the taxi driver*	*The "**operator**" should set the status of the hail to "`accepted_by_taxi`" by doing a "`PUT`" request on the /hails/{hail_id} API when the hail has been accepted by the taxi driver. Time before timeout_by_customer 10 minutes.*
`declined_by_taxi`	*The hail has been declined by the taxi driver*	*The "**operator**" should set the status of the hail to "`declined_by_taxi`" by doing a "`PUT`" request on the /hails/{hail_id} API when the hail has been rejected by the taxi driver.*
`timeout_taxi`	*The taxi driver did not accept nor reject the hail after 30s*	*The TXP changes the status to `timeout_taxi` automatically after 30s have passed since the status was set to `received_by_taxi`.*
`accepted_by_customer`	*The hail has been confirmed by the client*	*The "**search engine**" should set the status of the hail to "`accepted_by_customer`" by doing a "`PUT`" request on the /hails/{hail_id} API when the hail has been confirmed by the client or the search engine itself(if it consider it to be better for the user experience)*

		Time before failure 1 hour. *Warning: this confirmation can only happen "after" the status has been set to "`accepted_by_taxi`" by the "**operator**".*
`declined_by_customer`	*The hail has been canceled by the client*	*The "**search engine**" should set the status of the hail to "`declined_by_customer`" by doing a "`PUT`" request on the /hails/{hail_id} API when the hail has been canceled by the client.* *Warning: this cancellation can happen at any moment (including before the taxi driver accepts the hail).*
`timeout_customer`	*The client did not confirm nor cancel the hail after 10 minutes*	*The "TXP" changes the status to "`timeout_customer`" automatically after 10 minutes have passed since the status was set to `accepted_by_taxi`.*
`incident_customer`	*An event of force majeure prevents the client to wait for the taxi*	*The "**search engine**" should set the status of the hail to "`incident_customer`" by doing a "`PUT`" request on the /hails/{hail_id} API when the client cancels the hail after having reconfirmed it.*
`incident_taxi`	*An event of force majeure prevents the taxi to serve the client*	*The **operator** should set the status of the hail to "`incident_taxi`" by doing a "`PUT`" request on the /hails/{hail_id} API when the taxi cancels the hail after having accepted it.*
`failure`	*A technical problem happened.*	*The "TXP" changes the status to "`failure`" when:* • *The **operator** endpoint is unreachable;* • *or when receiving an HTTP 4xx or 5xx response from the **operator** endpoint;* • *or if the **operator** endpoint does not return a `hail` JSON object containing a valid `taxi_phone_number`;* • *or if the **operator** does not set the status of the `hail` to `received_by_taxi` in the 10s after the status has been set*

		to `received_by_operator`.
`customer_on_board`	*The customer is on board.*	*The "**operator**" should set the status of the hail to `customer_on_board` by doing a "`PUT`" request on the /hails/{hail_id} API when the hail has been accepted by the taxi driver. No timeout.*
`finished`	The hail is finished	*The "**operator**" should set the status of the hail to `finished` by doing a "`PUT`" request on the /hails/{hail_id} API when the hail has been accepted by the taxi driver.*

4. Hails simulation

4.1 Searching for a taxi

In the acceptation environment, the search will only return fake taxis from a fake operator (See 2.1.1). The fake taxis will be positioned randomly around the longitude and latitude sent as parameters. Each fake taxi will represent one of the test scenario described in section 4.3. For fake taxis, the attribute is not used to represent the model of the vehicle. Instead, the attribute model specifies which test scenario the fake taxi should be used for (ex: model='happy_path').
Some information about the taxi should be displayed on the application interface including the vehicle model before the client selects the taxi.

```
GET /api/taxis

Example: /api/taxis?lat=45.511885&lon=-73.607919

Parameters

query

lon                    (string) (required)

lat                    (string) (required)

count                  (integer) (required)
```

4.2 Hailing a fake taxi

Once you have a selection of fake taxis, you can select one and create a hail the same way as described in section 2.1.2. Note that the taxis have to be hailed within 5 minutes after which they become unavailable.

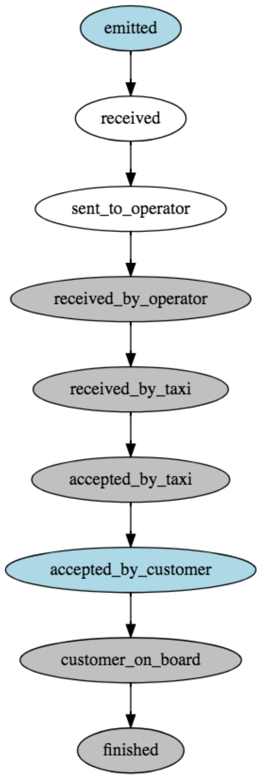
A fake operator is automatically generated for the search engine to simulate the updates normally made by the operator. The updates are done automatically upon receiving the hail and depending on the vehicle model the status will be updated to follow that scenario.

4.3 Tests Scenarios

A Search engine must support all the scenarios described in this section. In all scenarios, you must first fetch the taxi as described in 4.1 and select the taxi with the vehicle model corresponding to the test scenario to be executed.

4.3.1 Happy Path

4.3.1.1 Graphical representation



4.3.1.2 Details

- The client hails a taxi through the search engine.

```
POST /api/hails/
{
    "data": [
      {
        "customer_lat": 45.495,
        "customer_lon": -73.554,
        "customer_address": "70 Jarry",
        "taxi_id": "{taxiId}",
        "customer_phone_number": "514 201-4454",
        "operateur": "searchEngineEmail_test_operator",
       "customer_id": "anonymous"
      }
    ]
  }
```

- The operator receives a hail request from the TXP.
- Within 10 seconds, the operator notifies the TXP that they asked if the driver wanted to accept the hail. (status: “received_by_taxi”)
- When the driver accepts the hail within 30 seconds, the operator notifies the TXP. (status: “accepted_by_taxi”)
- The search engine need to fetch the status of the hail (preferably every 5 seconds) to check if the status changed.

```
GET /api/hails/{hailId}
```

- When the status is “accepted_by_taxi” the search engine must confirm the hail by sending the status “accepted_by_customer within 10 minutes.

```
PUT /api/hails/{hailId}
{"data":[{"status":"accepted_by_customer"}]}
```

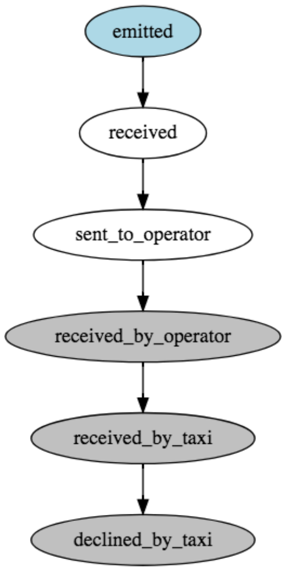
- The taxi then has 1 hour to pick up the client at it’s location.
- When the operator receives the information that the client is on board, the operator notifies the TXP. (status: “customer_on_board”)

- When the client leaves the taxi, the operator notifies the TXP. (status: “finished”)
- Once the hail is finished the search engine can propose the client to rate the ride from 1 to 5 (please refer to 2.1.2 Hailing a taxi for possible values) . Note that the evaluation could be done while the status is “customer_on_board”.

```
PUT /api/hails/{hailId}

{"data":[{

        "rating_ride": 0,

        "rating_ride_reason": "ko"

}]}
```

4.3.2 Not accepted by taxi

4.3.2.1 Graphical representation



4.3.2.2 Details

- The client hails a taxi through the search engine.

```
POST /api/hails/
{
    "data": [
      {
        "customer_lat": 45.495,
        "customer_lon": -73.554,
        "customer_address": "70 Jarry",
        "taxi_id": "{taxiId}",
        "customer_phone_number": "514 201-4454",
        "operateur": "searchEngineEmail_test_operator",
       "customer_id": "anonymous"
      }
    ]
  }
```

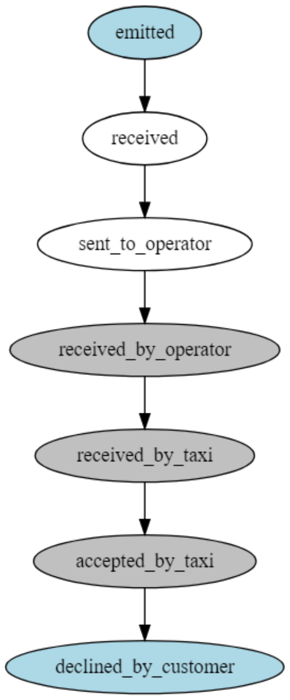
- The operator receives a hail request from the TXP.

- Within 10 seconds, the operator notifies the TXP that they asked if the driver wanted to accept the hail. (status: “received_by_taxi”)
- When the driver declined the hail within 30 seconds, the operator notifies the TXP. (status: “declined_by_taxi”)
- The search engine notifies the client.

```
GET /api/hails/{hailId}
```

4.3.3 Declined by customer

4.3.3.1 Graphical representation



4.3.3.2 Details

- The client hails a taxi through the search engine.

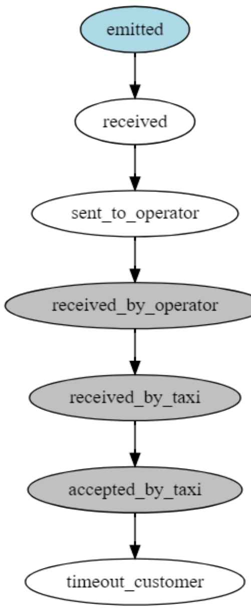
```
POST /api/hails/
{
    "data": [
      {
        "customer_lat": 45.495,
        "customer_lon": -73.554,
        "customer_address": "70 Jarry",
        "taxi_id": "{taxiId}",
        "customer_phone_number": "514 201-4454",
        "operateur": "searchEngineEmail_test_operator",
       "customer_id": "anonymous"
      }
    ]
  }
```

- The operator receives a hail request from the TXP.
- Within 10 seconds, the operator notifies the TXP that they asked if the driver wanted to accept the hail. (status: “received_by_taxi”)
- When the driver accepts the hail within 30 seconds, the operator notifies the TXP. (status: “accepted_by_taxi”)
- Then the customer decline the hail within 10 minutes, the search engine notifies the TXP.

```
PUT /api/hails/{hailId}

{"data":[{"status":"declined_by_customer"}]}
```

4.3.4 Accepted by customer after timeout

4.3.4.1 Graphical representation



4.3.3.2 Details

- The client hails a taxi through the search engine.

```
POST /api/hails/
{
    "data": [
      {
        "customer_lat": 45.495,
        "customer_lon": -73.554,
        "customer_address": "70 Jarry",
        "taxi_id": "{taxiId}",
        "customer_phone_number": "514 201-4454",
        "operateur": "searchEngineEmail_test_operator",
       "customer_id": "anonymous"
      }
    ]
  }
```

- The operator receives a hail request from the TXP.

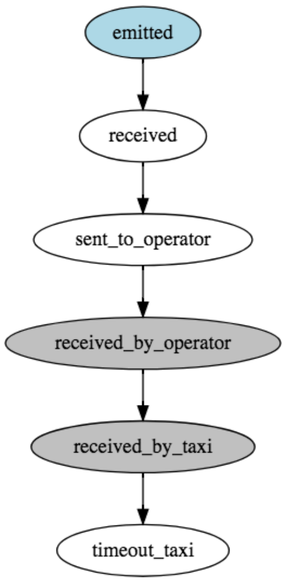
- Within 10 seconds, the operator notifies the TXP that they asked if the driver wanted to accept the hail. (status: “received_by_taxi”)
- When the driver accept the hail within 30 seconds, the operator notifies the TXP. (status: “accepted_by_taxi”)
- Then the customer accept the hail **after** 10 minutes, the search engine notifies the TXP.

```
PUT /api/hails/{hailId}

{"data":[{"status":"accepted_by_customer"}]}
```

- The status of the hail should now be “timeout_customer” and the search engine notify the client.

4.3.5 *Accepted by taxi after timeout*

4.3.5.1 Graphical representation



4.3.5.2 Details

- The client hails a taxi through the search engine.

```
POST /api/hails/
{
    "data": [
      {
        "customer_lat": 45.495,
        "customer_lon": -73.554,
        "customer_address": "70 Jarry",
        "taxi_id": "{taxiId}",
        "customer_phone_number": "514 201-4454",
        "operateur": "searchEngineEmail_test_operator",
       "customer_id": "anonymous"
      }
    ]
  }
```

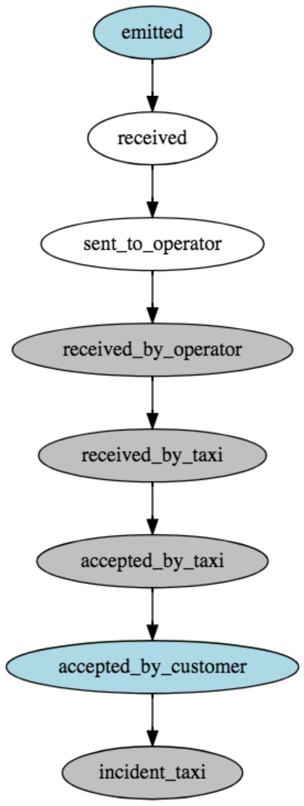
- The operator receives a hail request from the TXP.

- Within 10 seconds, the operator notifies the TXP that they asked if the driver wanted to accept the hail. (status: “received_by_taxi”)
- When the driver accepts the hail **after** 30 seconds, the operator notifies the TXP.
- The status of the hail should now be “timeout_taxi”.
- The search engine notifies the client.

```
GET /api/hails/{hailId}
```

4.3.6 *Cancelled by taxi after accepted by customer*

4.3.6.1 Graphical representation



4.3.6.2 Details

- The client hails a taxi through the search engine.

```
POST /api/hails/
{
    "data": [
      {
        "customer_lat": 45.495,
        "customer_lon": -73.554,
        "customer_address": "70 Jarry",
        "taxi_id": "{taxiId}",
        "customer_phone_number": "514 201-4454",
        "operateur": "searchEngineEmail_test_operator",
       "customer_id": "anonymous"
      }
    ]
```

```
  }
```

- The operator receives a hail request from the TXP.
- Within 10 seconds, the operator notifies the TXP that they asked if the driver wanted to accept the hail. (status: “received_by_taxi”)
- When the driver accepts the hail within 30 seconds, the operator notifies the TXP. (status: “accepted_by_taxi”)
- When the status is “accepted_by_taxi” the search engine must confirm the hail by sending the status “accepted_by_customer within 10 minutes.

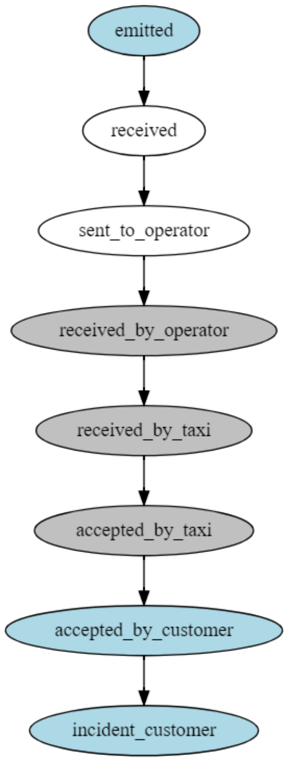
```
PUT /api/hails/{hailId}

{"data":[{"status":"accepted_by_customer"}]}
```

- When an incident occurs that prompted the driver to cancel the hail before having the client on board, the operator notifies the TXP and specifies the reason why the hail was canceled. (status: “incident_taxi”, incident_reason:”breakdowns”)
- The search engine notifies the client.

```
GET /api/hails/{hailId}
```

4.3.7 Cancelled by customer

4.3.7.1 Graphical representation



4.3.7.2 Graphical represent

- The client hails a taxi through the search engine.

```
POST /api/hails/
{
    "data": [
      {
        "customer_lat": 45.495,
        "customer_lon": -73.554,
        "customer_address": "70 Jarry",
        "taxi_id": "{taxiId}",
        "customer_phone_number": "514 201-4454",
        "operateur": "searchEngineEmail_test_operator",
       "customer_id": "anonymous"
      }
    ]
```

```
  }
```

- The operator receives a hail request from the TXP.
- Within 10 seconds, the operator notifies the TXP that they asked if the driver wanted to accept the hail.
- When the driver accepts the hail within 30 seconds, the operator notifies the TXP.
- When the client accepts the hail within 10 minutes, the search engine notifies the TXP.

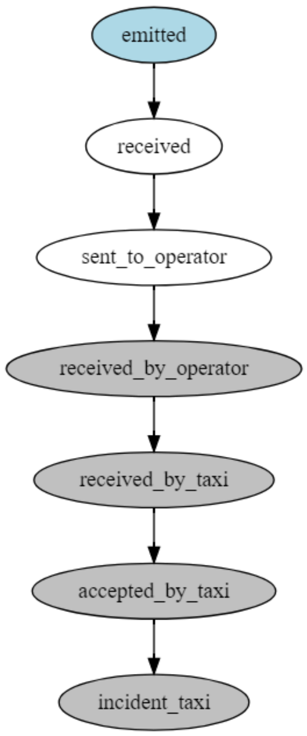
```
PUT /api/hails/{hailId}

{"data":[{"status":"accepted_by_customer"}]}
```

- When an incident occurs that prompted the client to cancel the hail before being on board (please refer to 2.1.2 Hailing a taxi for possible values), the search engine notifies the TXP.

```
PUT /api/hails/{hailId}

{"data":[{"status":"incident_customer",

          "incident_customer_reason":""

}]}
```

4.3.8 Cancelled by taxi before accepted by customer

4.3.8.1 Graphical representation



4.3.8.2 Details

- The client hails a taxi through the search engine.

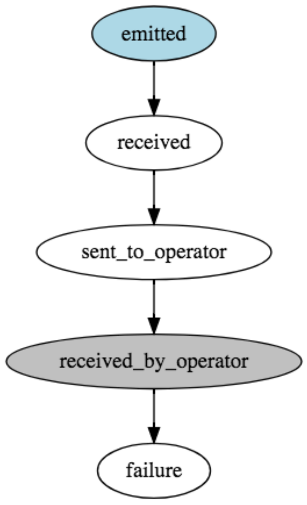
```
POST /api/hails/
{
    "data": [
      {
        "customer_lat": 45.495,
        "customer_lon": -73.554,
        "customer_address": "70 Jarry",
        "taxi_id": "{taxiId}",
        "customer_phone_number": "514 201-4454",
        "operateur": "searchEngineEmail_test_operator",
       "customer_id": "anonymous"
      }
    ]
  }
```

- The operator receives a hail request from the TXP.
- Within 10 seconds, the operator notifies the TXP that they asked if the driver wanted to accept the hail. (status: “received_by_taxi”)
- When the driver accepts the hail within 30 seconds, the operator notifies the TXP. (status: “accepted_by_taxi”)
- When an incident occurs that prompted the driver to cancel the hail before having the client on board, the operator notifies the TXP and specifies the reason why the hail was cancel. (status: “incident_taxi”, incident_taxi_reason: “breakdown”)
- The search engine notify the client.

```
GET /api/hails/{hailId}
```

4.3.9 *Failure example*

4.3.9.1 Graphical representation



4.3.9.2 Details

- The client hails a taxi through the search engine.

```
POST /api/hails/
{
    "data": [
      {
        "customer_lat": 45.495,
        "customer_lon": -73.554,
        "customer_address": "70 Jarry",
        "taxi_id": "{taxiId}",
        "customer_phone_number": "514 201-4454",
        "operateur": "searchEngineEmail_test_operator",
       "customer_id": "anonymous"
      }
    ]
  }
```

- The operator receives a hail request from the TXP.
- After 10 seconds, the operator notifies the TXP that they asked if the driver wanted to accept the hail. (status: “received_by_taxi”)
- If nothing happen after this point, the status should be set to failure and retrieve with the GET method. The search engine

should poll the TXP until it get the “failure” status and then notify the client.

```
GET /api/hails/{hailId}
```

4.3.10 Unexpected Exceptions

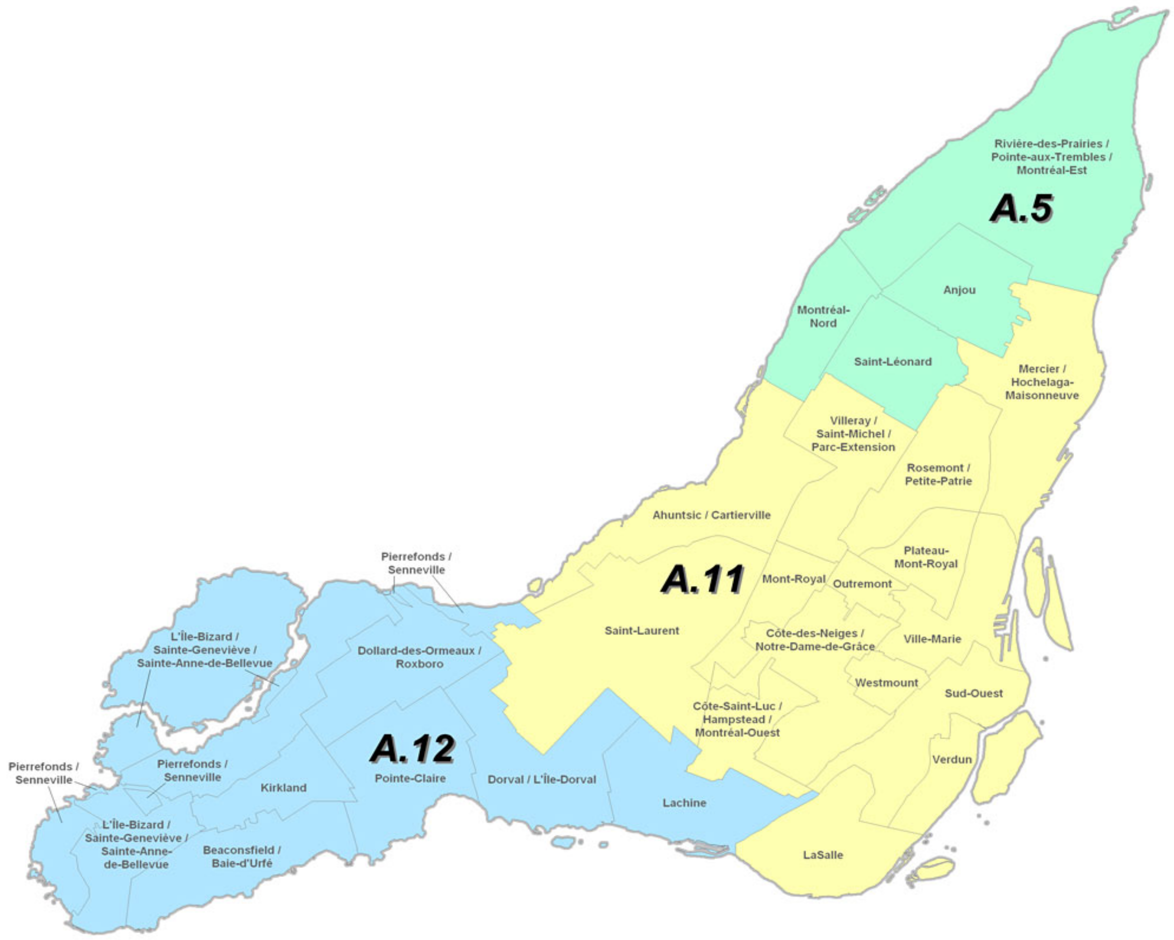
A state transition to the state failure may occur from any state. Moreover, technical problems may cause the TXP to be unreachable or to respond with an HTTP 500 status code (server error). These are edge cases that should not occur frequently, but the search engine must be ready to deal with these situations. If an unexpected error occurs, the search engine should assume that the state of the hail is failure and should stop interacting with the TXP for this hail.

The search engine must notify the client that a technical problem occurred and that the hail is canceled if the technical problem occurred.

5. Other references

5.1 Taxi zones (ZUPC)

- 102005: A5 – Eastern part of the island of Montreal
- 102011: A11 – Downtown/center Montreal
- 102012: A12- West part of the island of MontrealA.5
- Montréal Pierre Elliott Trudeau Airport (Excluded)


Rivière-des-Prairies / Pointe-aux-Trembles / Montréal-Est
A.5
Anjou
Montréal-Nord
Saint-Léonard
Mercier / Hochelaga-Maisonneuve
Villeray / Saint-Michel / Parc-Extension
Rosemont / Petite-Patrie
Ahuntsic / Cartierville
Plateau-Mont-Royal
A.11
Mont-Royal
Outremont
Saint-Laurent
Côte-des-Neiges / Notre-Dame-de-Grâce
Ville-Marie
Westmount
Sud-Ouest
Côte-Saint-Luc / Hampstead / Montréal-Ouest
Verdun
LaSalle
Lachine
Pierrefonds / Senneville
L'Île-Bizard / Sainte-Geneviève / Sainte-Anne-de-Bellevue
Dollard-des-Ormeaux / Roxboro
A.12
Pointe-Claire
Dorval / L'Île-Dorval
Pierrefonds / Senneville
Pierrefonds / Senneville
Kirkland
L'Île-Bizard / Sainte-Geneviève / Sainte-Anne-de-Bellevue
Beaconsfield / Baie-d'Urfé