D1.1: Big travel demand data analytics support tool

Project acronym:	TRANS-FORM
Project title:	Smart transfers through unravelling urban form and travel flow dynamics
Funding Scheme:	ERA-NET call on Smart Cities and Communities (ENSCC)

Authors

Erick Bastidas	ebas@ch.ibm.com
Marco Laumanns	mlm@zurich.ibm.com
Thomas Pfeiffer	thomas.pfeiffer@de.ibm.com
Joelle Tavernier	joelle.tavernier@ch.ibm.com

Internal Reviewer

Flurin Hänseler	F.S.Hanseler@tudelft.nl

State:	Version 1, revised after internal review
Distribution:	Confidential
Date:	August 8, 2017

Deliverable History

Date	Author	Changes
30-01-2017 09-02-2017 01-03-2017 22-03-2017	Erick Bastidas Erick Bastidas Erick Bastidas Joelle Tavernier	Initial draft Swedish, Dutch, Swiss user stories and initial API endpoints draft Swedish, Dutch, Swiss API endpoints final draft and described in the Swagger file Revision of API definitions and examples
09-04-2017	Marco Laumanns	Section 2 added, Overall revision
29-06-2017	Erick Bastidas	Revision of API definitions stop-to-stop/average
04-08-2017	Marco Laumanns	Revision based on internal review
08-08-2017	Marco Laumanns	Adjustment of API specifications to current implementations. Added Appendix A to reflect the changes

Contents

1. Summary

Task 1.1 deals with the design and development of a tool to process, filter and analyze travel demand data. In several iterations of the requirements analysis process, the available data sets from the case studies were analyzed and the desired functionalities of the data analysis tool were defined together with all partners. Based on the requirements, the architecture of the tool was designed, and the tool was implemented accordingly.

The data storage backend consists of a modern graph database, which contains a transit layer and a passenger layer to store the cleaned and preprocessed input data. The architecture of the tool and the data model is described on Section 2.

Queries against the database are implemented in the Apache Gremlin open source graph query language and can be called via a set of well-defined REST APIs. The APIs are defined using Swagger 2.0, which ensures easy documentation and integration into other applications. The Swagger file can be downloaded from the TRANS-FORM shared folder at Box:

- TRANS-FORM > WP1 > Task 1.1 > Application Design > APIs > swagger-api.yaml
- TRANS-FORM > WP1 > Task 1.1 > Application Design > APIs > swagger-api.json

Besides the main data analysis functionality, the tool also provides administrative features such as user authentication and lower-level data inspection. These administrative APIs are described in Section 3. The functional APIs for running the data analysis queries as well as the User Stories that led to their definition, are described in detail in Section 4, which forms the major part of this document.

Overall, the tool provides APIs for the following four types of evaluations (and corresponding endpoints):

- OD matrices, which contain the (total and time averaged) number of “passenger journeys” between given “stops”
  - */compute/passengers/stop-to-stop/count*
  - */compute/passengers/stop-to-stop/average*
- Transfer matrices, which contain the number of passengers transfering between different “trips” or “routes” (groups of trips of the same line):
  - */compute/passengers/transfer/trip-to-trip/count*
  - */compute/passengers/transfer/trip-to-trip/waiting-time*
  - */compute/passengers/transfer/route-to-route/count*
  - */compute/passengers/transfer/route-to-route/average*
  - */compute/passengers/transfer/route-to-route/waiting-time/distribution*

- Intra-station passenger zone-to-zone matrices, which contain the number of passengers and their walking time, speed and distances between given zones inside the station
    - */compute/passengers/intra-station/zone-to-zone/count*
    - */compute/passengers/intra-station/zone-to-zone/average*
    - */compute/passengers/intra-station/zone-to-zone/walking-distance/distribution*
    - */compute/passengers/intra-station/zone-to-zone/walking-speed/distribution*
    - */compute/passengers/intra-station/zone-to-zone/walking-time/distribution*
- Intra-station passenger density and walking speed distribution (for a given space-time discretization)
    - */compute/passengers/intra-station/zone-to-zone/density*
    - */compute/passengers/intra-station/zone-to-zone/walking-speed/average/density*

Section 5 concludes this document by discussing potential future enhancements and well as deployment options.

2. Data Model and Application Architecture

Analyzing demand data of public transportation systems requires the linkage of two different kinds of data: data related to public transit vehicles (such as vehicle trips, routes, stops and schedules) and data related to the travel flows (such as smart card, video data or other data capturing the journeys of individual travelers). Accordingly, the data model of the application is designed to reflect this dichotomy and consists of the following two layers:

- The *Transit Layer* contains the definitions of the transit services information. For this layer the data model follows the de-facto standard GTFS. The GTFS ("General Transit Feed Specification") defines a common format for public transportation schedules and associated geographic information. A GTFS "feed" is composed of a series of text files (in csv format), usually collected in a ZIP file. Each file models a particular aspect of transit information: stops, routes, trips, and other schedule data.
- The *Passenger Layer* contains definitions of the passengers' movements when using the transit services or when transferring between services. In this project, this data is obtained from automatic fare collection (AFC) systems as well as video data.

The purpose of the data model is to store and to link these two kinds of data in order to facilitate complex analytics to be run against it as efficiently as possible.

2.1. Transit Layer

The data model components for the Transit Layer reflect the consensus between the project partners to rely as much as possible on existing data standards, in this case GTFS. In order to maintain compatibility, all the terminology (file names and field names in each file) must be used as the GTFS specifies. Moreover, the GTFS minimum "required" files and fields must be provided when adding a new transportation service. The main GTFS entities that are used in our applications are as follows:

- A *stop* is a location that serves as an access point to the transit system for passengers.
- A *stop time* represents a stopping event of a public transport vehicle at a stop in order to let passengers board the vehicle or disembark. It usually contains an arrival and a departure time as properties.
- A *trip* represents the movement of a vehicle along stops. It is represented by a sequence of stop time entities.
- A *route* is a collection of different trips, e.g., to group the different trips that visit the same sequence of stops.
- A *calendar* is a temporal pattern in terms of days of operation that can be associated with a trip.

- An *agency* is the transport operator of a route.

The concept of a “line”, which is common in the public transport literature, does not exist as such in the GTFS, but can be represented as a *route*.

To extend the TRANS-FORM Transit Layer data specification to include new objects that are not part of the GTFS, two approaches can be considered:

- to create a new “.txt” file and add the new fields to this new file, including the connections (foreign ids) to the standard GTFS files; or
- to add “optional” fields to a GTFS file, if the data relates directly to a GTFS object/file.

The first method is always preferred to keep GTFS intact.

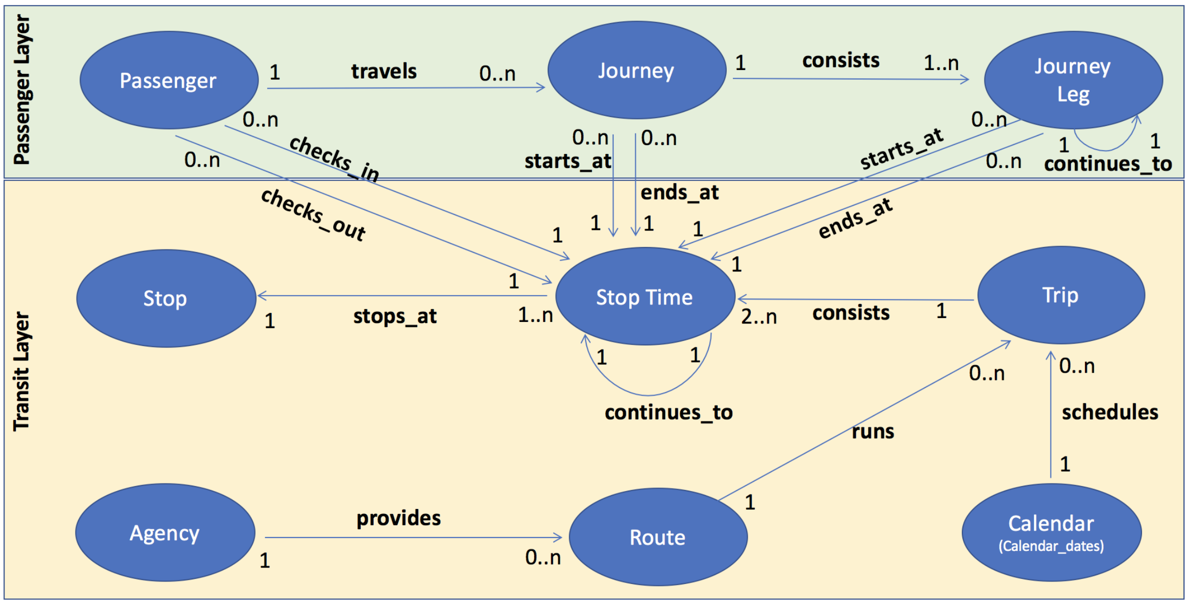
2.2. Passenger Layer

In contrast to the Transit Layer, there seems to be no standard data format available to describe passenger journeys. For this reason a new data model for the Passenger Layer is defined from scratch for the purpose of this application.

In order to define the Passenger Layer data model, a few fundamental concepts have to be defined:

- A *passenger* is a unique person who is doing one of multiple *journeys* over a given time frame.
- A *journey* reflects the use of a public transport system for a certain purpose, from an origin to a destination. Origin and destination refer to public transit *stop* objects in the Transit Layer. A *journey* is a path of one or *more journey legs*. (NB: the term *trip* is reserved for the Transit Layer, where it refers to a vehicle trip).
- A *journey leg* refers to the usage of a particular vehicle *trip* from a boarding *stop* to an alighting *stop*.

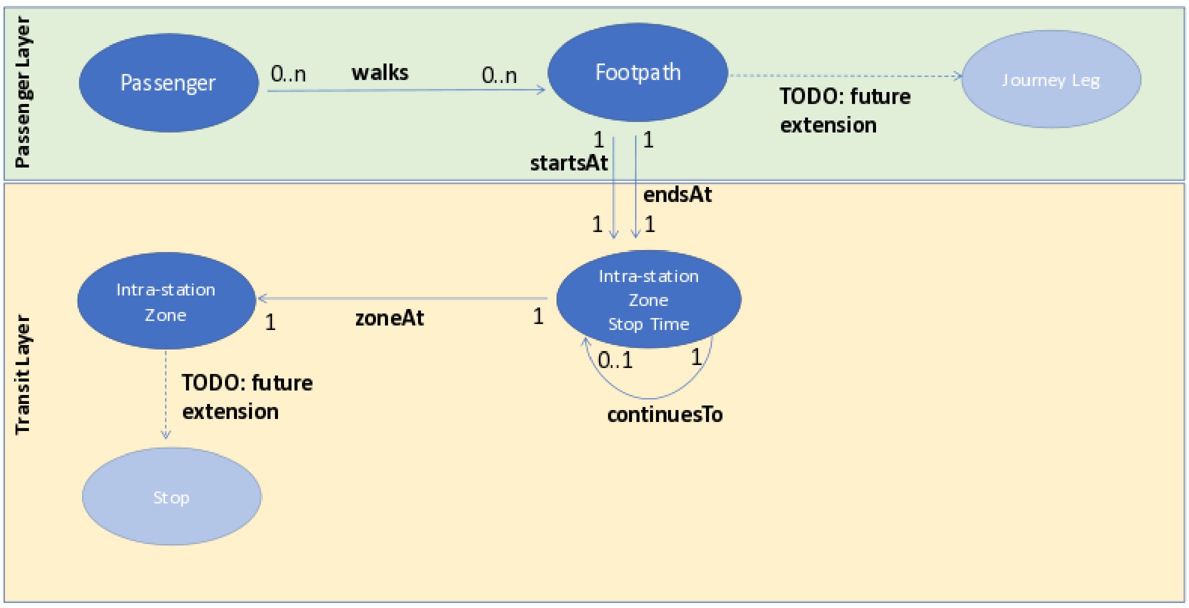
The data model across the two layers is shown schematically in the following diagram:



This is a Graph-based data model, which is the underlying structure of a Graph Database as explained below in Section 2.4. The numbers at the heads and tails of the edges (blue arrows) indicate whether the edge represents a one-to-one, a one-to-many, many-to-one or a many-to-many relationship, similar as in standard entity-relationship-diagrams. For example, a trip consists of many (but at least 2) stop time objects, while a stop time object can be associated to exactly one trip.

2.3. Extensions for hub-level data

The purpose of the hub extension is to store, at an appropriate level of detail, traces of the passenger behavior inside the transportation hubs. Instead of modeling it by adding a third layer to the data model, it is implemented as an extension for the transit and passenger layers by adding further objects and relations. This design decision is based on the fact that the hub layer adds some further detailed data (here: footpaths) about an already defined object (the passenger) and associates those to objects of the Transit Layer (the stops). The following diagram shows the hub-level extensions of the data model:



2.4. Graph Database

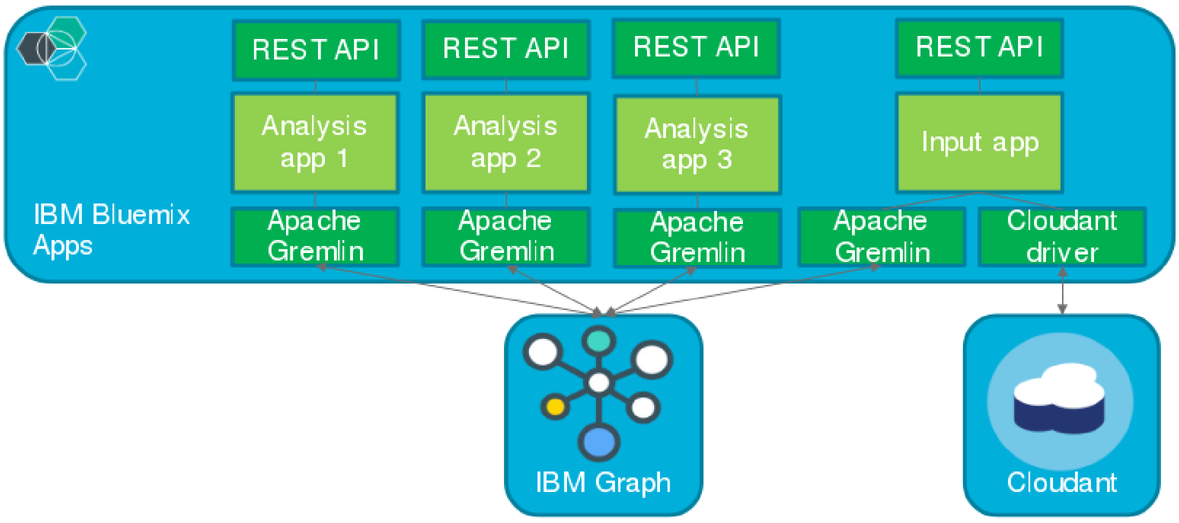
The application data model sketched above consists of data objects and their relations. This structure can nicely be reflected in a graph database. Graph databases are a relatively recent concept that store relationsships between individual data objects directly instead of using tables with foreign keys as in traditional relational databases. This should lead to faster processing times for complex queries by avoiding the need to execute multiple nested “join” operations on tables in a relational database.

In a graph database, the data objects are called “vertices” and the relations between two vertices are called “edges”. Both vertices and edges can be of a certain kind (called “label”) and can store internal data (called “properties”). Edges have an orientation, so each edge has a unique “out-vertex” and a unique “in-vertex”. The orientation can be used with queries, e.g., to filter on only outgoing or only incoming edges, but they can be traversed in both directions with a query.

In order to retrieve data from a graph database, a graph-oriented query language is typically used instead of the common SQL used for relational databases. In this application, the graph programming language Gremlin is used, which is part of the Apache TinkerPop open source project.

2.5. Architecture

The following diagram gives a high-level overview of the application architecture:



The application is designed as a web application running on IBM Bluemix, a Platform-as-a-Service (PaaS) built on Cloud Foundry, an open source, multi cloud application platform as a service. The data backend of the application is IBM Graph, a Database-as-a-Service (DBaaS) running on IBM Bluemix. IBM Graph is a graph database with support for Gremlin and implements the data model described above.

The core of the application consists of a series of light-weight “analysis apps”, which are serving the REST APIs by running queries against the graph database using Gremlin. These apps are implemented in JavaScript running on a Node.js runtime environment. However, the design is flexible such that further apps can be added while being implemented in any arbitrary programming language and framework that is able to serve REST APIs and access a graph database via Gremlin queries.

In order to load data into the main graph database, some custom Java applications are implemented that can be executed offline. The intention is that the application's administrator performs all data loads offline using these applications.

The analysis functionality, which is the main functionality of the tool, is provided via a number of REST APIs, which are described in detail in the following two sections.

3. Internal API endpoints (core)

This section describes the “internal” API endpoints. The purpose of the internal endpoints is not to perform the actual data analysis but to provide administration capabilities, data management and support for developers. The internal APIs are grouped into four categories:

- Auth: APIs related to user authentication
- User: APIs for users to manage their user profile
- Admin: APIs for administrators to manage user accounts
- CRUD: low-level APIs to inspect and edit the data (vertices and edges) of the underlying graph database directly.

3.1. Auth: Setting up the users authentication api

```
/api/v1/auth/signup
  POST: users.signup

/api/v1/auth/signin
  POST: users.signin

/api/v1/auth/signout
  GET: users.signout

/api/v1/auth/forgot
  POST: users.forgot

/api/v1/auth/reset/:token
  GET: users.validateResetToken

/api/v1/auth/reset/:token
  POST: users.reset
```

3.2. Users: Setting up the users profile api

```
/api/v1/users/me
  GET: users.me

/api/v1/users
  PUT: users.update

/api/v1/users/accounts
  DELETE: users.removeOAuthProvider

/api/v1/users/password
  POST: users.changePassword

/api/v1/users/picture
  POST: users.changeProfilePicture
```

3.3. Admin: Setting up the admin user-management api

```
/api/v1/users
  GET: adminOnly

/api/v1/users/:userId
  GET: adminOnly, users.read
  PUT: adminOnly, users.update
  DELETE: adminOnly, users.delete
```

4. External API endpoints (data computation)

The external API endpoints are those that provide access to the actual data analysis queries and computations. The APIs have been developed to fulfill the analysis requirements from all project partners related to the case studies. This section first provides the user stories that were formulated during the requirements elicitation process and the next section then defines and describes the implemented APIs in detail.

4.1. User Stories

In order to specify the requirements, the software development concept of user stories was applied. User stories are brief, informal descriptions (in natural language) of features that a software should exhibit in order to fulfill a certain requirement for an end user. For the purpose of this project the end user is a transportation analyst who wants to analyse transport demand data. In the following three subsections, the user stories are listed, grouped according to the respective level (urban level, regional level and hub level). Reference to data examples (input and output) are given.

4.1.1. Urban Level User Stories

1. *USER STORY: "As a public transport analyst, I want to get a stop-to-stop OD-matrix of passenger journeys, as average over the number of imported days, cleaned, matched and corrected for non-smartcard users."*

   Check:
   - API endpoint: /api/v1/compute/passengers/stop-to-stop/count
   - API endpoint: /api/v1/compute/passengers/stop-to-stop/average

   Use:
   - average_by: TOTAL_DAYS_OF_TIME_INTERVAL

   Examples: 1, 5, 6, 7 and others

2. *USER STORY: " As a public transport analyst, I want to get a public stop name–to–public stop name OD-matrix on a passenger journey level, as average over the number of imported days, cleaned, matched and corrected for non-smartcard users."*

   Check:
   - API endpoint: /api/v1/compute/passengers/stop-to-stop/count
   - API endpoint: /api/v1/compute/passengers/stop-to-stop/average

   Use:
   - average_by: TOTAL_DAYS_OF_TIME_INTERVAL,

- use “stop_name” instead of “stop_id” to get public stop names

Examples: 5, 6, 7 and others

3. *USER STORY: “As a public transport analyst, I want to get a time-dependent OD-matrix (per hour of the day / other time aggregation level)”*

   Check:
   - API endpoint: /api/v1/compute/passengers/stop-to-stop/count
   - API endpoint: /api/v1/compute/passengers/stop-to-stop/average

   Use:
   - average_by: PER_HOUR_OF_DAY

   Examples: 5, 6 and others

4. *USER STORY: “As a public transport analyst, I want to get a segmented OD-matrix for several trip purposes / classes”*

   This user story has not been implemented, since the available data has no information on trip purposes or class.

4.1.2. Regional Level User Stories

1. *USER STORY: "As a public transport analyst, I want to get a station-specific transfer matrix that contains the number of transfering passengers between given pairs of vehicle trips."*

   Check:
   - API endpoint: /api/v1/compute/passengers/transfer/trip-to-trip/count
   - API endpoint: /api/v1/compute/passengers/transfer/trip-to-trip/waiting-time

   Examples: 1, 2

2. *USER STORY: "As a public transport analyst, I want to get a station-specific transfer matrix that contains the number of transfering passengers between given pairs of vehicle routes as well as information about the transfer time (such as average and distributions of transfer time)."*

   Check:
   - API endpoint: /api/v1/compute/passengers/transfer/route-to-route/count
   - API endpoint: /api/v1/compute/passengers/transfer/route-to-route/average
   - API endpoint: /api/v1/compute/passengers/transfer/waiting-time/distribution

   Use:
   - average_by: TOTAL_DAYS_OF_TIME_INTERVAL
   - "distribution_percentile_interval" to specify the resolution of the discretized transfer time distribution

   Examples: 1, 2

4.1.3. Hub Level User Stories

1. *USER STORY: For a given intra-station pair of origin and destination zone, each specified by a list of geo-coordinates (the corner points of the rectangle descibing the physical area), I want to know the number of passengers who walked between those zones in a given time interval.*
    a. *Same as (1), just for a given list of intra-station OD zone pairs.*

    Check:
    - API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/count
    - API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/average

    Use:
    - “od_input_type: ZONE_LIST”
    - average_by: TOTAL_DAYS_OF_TIME_INTERVAL
    - average_by: PER_HOUR_OF_DAY

    See Examples: 1, 2

2. *USER STORY: For a given intra-station pair of origin and destination zone, I want to know the distribution of walking time [in seconds] of passengers who walked between those zones, in a given time interval. I'm expecting to receive the (cumulative) distribution as a list of percentiles in default 5% increments, that is, values of the distribution at the 5%, 10%, 15%, ..., 95%.*
    a. *Same as (2) but instead of the default distribution increments, I want to specify a list of percentile and expect that list to be filled with the corresponding values.*

    Check:
    - API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/walking-time/distribution

    Use:
    - use “od_input_type: ZONE_LIST”
    - use “distribution_percentile_interval” to specify the resolution of the discretized distribution

    See Examples: 3

3. *USER STORY: Same as (2) but I want to know the walking distance distribution instead of the walking time.*

    Check:

- API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/walking-distance/distribution

Use:

- use “od_input_type: ZONE_LIST”
- use “distribution_percentile_interval” to specify the resolution of the discretized distribution

See Examples: 3

4. *USER STORY: Same as (2) but I want to know the speed distribution instead of the walking time, where velocity [in mm/s] for a passenger is walking_distance / walking_time.*

   Check:

   - API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/walking-speed/distribution

   Use:

   - use “od_input_type: ZONE_LIST”
   - use “distribution_percentile_interval” to specify the resolution of the discretized distribution

   See Examples: 3

5. *USER STORY: For a given cell discretization (a rectangular grid specified by its origin point and a distance resolution in millimeters), I want to know the density (= number of passengers per m^2) in each grid cell in each time interval (e.g. every 30 seconds) over a given time range (e.g. 8:00am-10:am on January 27).*

   Check:

   - API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/density

   Use:

   - “od_input_type: GRID”
   - "grid_details": { "grid_boundaries": { "corner": { "x" : 0, "y" : 0}, "length": 4000, "width" : 4000 }
   - “group_by_time_seconds: 30”

   See Examples: 4

6. *USER STORY: Same as (5), but I want to know the average walking speed in each grid cell in each time interval*

   Check:

- API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/walking-speed/average/density

Use:

- use “od_input_type: GRID”,
- "grid_details": { "grid_boundaries": { "corner": { "x" : 0, "y" : 0}, "length": 4000, "width" : 4000 }
- “group_by_time_seconds: 30”

See Examples: 4

4.2. Graph transversal computations

This subsection describes the data analysis APIs in detail. For each API, a textual description is given first, followed by its formal definition (input and output). Examples are given to illustrate the input and output

The APIs can be called at the following URL:

- host: travel-analytics.eu-gb.mybluemix.net
- basePath: /api/v1/

4.2.1. API endpoint: /api/v1/compute/passengers/stop-to-stop/count

NOTE: To avoid a lot of duplication of text when requests/responses of different APIs are very similar,, certain options as well as the corresponding API will be marked with a background color.

Similar API with additional options (marked in color):

- API endpoint /api/v1/compute/passengers/stop-to-stop/average

Description:
Returns an OD-Matrix with the (total or average) number of passengers (i.e. distinct passenger journeys), who travel (i.e., start their journey) within the time frame specified in the request, from (one or multiple) origin stops to (one or multiple) destination stops. The output OD-Matrix shows the count only from origin and destination of the complete journeys, i.e. the computation doesn't check the values per each intermediate journey legs (transfers) of a journey, but only from the origin and destination of the complete journey regardless of the route taken to complete it. For example, the OD-Matrix from Amsterdam to Zurich in a certain day, would count the number of distinct journeys with origin stop "Amsterdam Centraal stop_id_XYZ" and destination stop "Zurich HB stop_id_ABC", and not aggregate the count of each the individual possible journey legs (for example Amsterdam - Dusseldorf - Frankfurt - Basel - Zurich) nor consider the multiple different possible routes that passengers could take.

Computation outputs

- total_count
  - Endpoint: /api/v1/compute/passengers/stop-to-stop/count
  - Description: returns the total sum of all the journeys,
- average
  - Endpoint: /api/v1/compute/passengers/stop-to-stop/average
  - Description: Returns the average with respect to the specification of the "average_by" attribute of the request; the average computation is done after

the group_by_time computation, i.e., the average is valid within each group time or OD-Matrix.

<u>HTTP Verb:</u>
POST

<u>Input (send JSON request):</u>

default_country: (required) CH (Switzerland), SE (Sweden), NL (Netherlands),

time_interval:
- ***start_timestamp***: Date (required),
- ***end_timestamp***: Date (required),

origin_stops: (Array, optional)
- ***country***: (optional, overrides "default_country" only for this element) CH (Switzerland), SE (Sweden), NL (Netherlands),
- ***stop_id*** (required if stop_name is empty, unique value in database),
- ***stop_name:*** (required if stop_id is empty, NOT unique value in database),
- ***pairs_to***: (optional)
  - LIST_OF_STOPS (Default, pair this origin stop_id to each of the stop_id in the "destination_stops" attribute of the request);
  - ALL_POSSIBLE_CONNECTIONS (pair this origin stop to all the destination stops of every journey found in the database. By default, the output uses the same stop attribute label of this element: whether "id" or "name");
- ***aggregate_as_station*** (optional): FALSE (Default, count the outgoing journeys only from this stop); TRUE (the computation will find all of the children stops of this station-stop and aggregate all the journeys from its children stops to one single output. The stop node must have to the property "location_type=1", which refers to stations. Note from GTFS: a station can't have a parent station),

destination_stops: (Array, optional)
- ***country***: (optional, overrides "default_country" only for this element) CH (Switzerland), SE (Sweden), NL (Netherlands),
- ***stop_id*** (required if stop_name is empty, unique value in database),
- ***stop_name:*** (required if stop_id is empty, NOT unique value in database),
- ***pairs_to*** (optional)
  - LIST_OF_STOPS (Default, pair this destination stop_id to each of the stop_id in the "origin_stops" attribute of the request);
  - ALL_POSSIBLE_CONNECTIONS (pair this destination stop to all the origin stops of every journey found in the graph. By default, the output uses the same stop attribute label of this element: whether "id" or "name"),
- ***aggregate_as_station*** (optional): FALSE (Default, count the incoming journeys only from this stop); TRUE (the computation will find all of the children stops of this station-stop and aggregate all the journeys from its children stops to one single output. The stop node must have to the property

"location_type=1", which refers to stations. Note from GTFS: a station can't have a parent station),

aggregate_all_origin_stops_as_stations (optional): FALSE (Default, count the outgoing journeys from only and each stop listed in the "origin_stops" attribute of the request); TRUE (for each stop in the "origin_stops", the computation will find all of the children stops of this station stop and aggregate all the journeys from its children stops to one single output. The stop node must have to the property "location_type=1", which refers to stations. Note from GTFS: a station can't have a parent station),

aggregate_all_destination_stops_as_stations (optional): FALSE (Default, count the incoming journeys from only and each stop listed in the "destination_stops" attribute of the request); TRUE (for each stop in the "destination_stops", the computation will find all of the children stops of this station stop and aggregate all the journeys from its children stops to one single output count. The stop node must have to the property "location_type=1", which refers to stations. Note from GTFS: a station can't have a parent station),

average_by (only for endpoint /compute/passengers/stop-to-stop/average):

PER_HOUR_OF_DAY, is the average number of passengers per hour of day (00:00:00 - 00:59:59, 01:00:00 - 01:59:59, …, 23:00:00 - 23:59:59);
TOTAL_HOURS_OF_TIME_INTERVAL: total count of passenger journeys divided by the number of hours in between the end_timestamp and start_timestamp;
TOTAL_DAYS_OF_TIME_INTERVAL, Default, total count of passenger journeys divided by the number of days in between the end_timestamp and start_timestamp;
TOTAL_MINUTES_OF_TIME_INTERVAL
TOTAL_SECONDS_OF_TIME_INTERVAL

Output (expect JSON response):

request:

ALL THE FIELDS IN THE REQUEST

journeys: (Array If specified, one per time slice defined by the "average_by" value, in between the total time_interval; if none of the above is specified, one for the complete time_interval)

time_interval:

start_timestamp: Date (required),
end_timestamp: Date (required),

average_interval_value: STRING (only for endpoint /api/v1/compute/passengers/stop-to-stop/average

- For the type of PER_X_OF_Y:
  - "01:00:00 - 01:59:59" [if PER_HOUR_OF_DAY],
- For the type of TOTAL_X_PER_TIME_INTERVAL:
  - "17" [if TOTAL_HOURS_PER_TIME_INTERVAL, in between "2016-10-03 06:00:00" and "2016-10-03 23:00:00")

  - "61" [if TOTAL_DAYS_PER_TIME_INTERVAL, in between 01-nov to 31-dec])

od_matrix: (Array)

origin_stop_id: STRING (only appear depending on the value of "od_matrix_output"),
origin_stop_name: STRING (only appear depending on the value of "od_matrix_output"),
origin_stop_country (only appears if origin_stops[country] is not empty): CH (Switzerland), SE (Sweden), NL (Netherlands),
destination_stop_id: STRING (only appears depending on the value of "od_matrix_output"),
destination_stop_name: STRING (only appears depending on the value of "od_matrix_output"),
destination_stop_country (only appears if destination_stops[country] is not empty): CH (Switzerland), SE (Sweden), NL (Netherlands),
total_count: Integer (only appears depending on the value of "count_value_output")
average: Double (only appears depending on the value of "count_value_output")

metadata:

api_version: v1

Example 1 [single-day & single origin-destination]: I want to know the total number of passenger going from 'Zurich-Altstetten' (= stop1) to 'Zurich-Oerlikon' (= stop2) on January 26.

<u>Input:</u>

```
{
        "default_country": "CH",
        "time_interval":{
                "start_timestamp": "2017-01-26 00:00:00",
                "end_timestamp": "2017-01-26 23:59:59"
        },

        "origin_stops": [
                {
                         "stop_id": "stop1"
                }
        ],
        "destination_stops": [
                {
                         "stop_id": "stop2"
                }
```

```
    ]
}
```

Output:

```
{
    “metadata”:
    {
        “api_version”: “v1”
    },
    “request”: {
        “default_country”: “CH”,
        "time_interval":{
            “start_timestamp”: “2017-01-26 00:00:00”,
            “end_timestamp”: “2017-01-26 23:59:59”
        },
        “origin_stops”: [
            {
                “stop_id”: “stop1”
            }
        ],
        “destination_stops”: [
            {
                “stop_id”: “stop2”
            }
        ]
    },
    “journeys”: [
        {
            "time_interval":{
                “start_timestamp”: “2017-01-26 00:00:00”,
                “end_timestamp”: “2017-01-26 23:59:59”
            },
             “od_matrix”: [
```

origin_stop_id	**destination_stop_id**	**total_count**
stop1	stop2	5203

```
            ]
        }
    ]
}
```

Example 2 [single-day & single origin & all destinations]: I want to know the total number of passenger going from 'Zurich-Altstetten' (= stop1) to their stop destinations on January 26.

Input:

```
{
        “default_country”: “CH”,
        "time_interval":{
                “start_timestamp”: “2017-01-26 00:00:00”,
                “end_timestamp”: “2017-01-26 23:59:59”
        },
         “origin_stops”: [
                {
                        “stop_id”: “stop1”,
                        “pairs_to” “ALL_POSSIBLE_CONNECTIONS”
                }
        ]
}
```

Output:

```
{
        “metadata”:
        {
                “api_version”: “v1”
        },
        “request”: {
                “default_country”: “CH”,
                "time_interval":{
                        “start_timestamp”: “2017-01-26 00:00:00”,
                        “end_timestamp”: “2017-01-26 23:59:59”
                },
                 “origin_stops”: [
                        {
                                 “stop_id”: “stop1”,
                                “pairs_to” “ALL_POSSIBLE_CONNECTIONS”

                        }
                ]
        },
        “journeys”: [
                {
                        "time_interval":{
                                “start_timestamp”: “2017-01-26 00:00:00”,
                                “end_timestamp”: “2017-01-26 23:59:59”
                        },
                         “od_matrix”: [
```

origin_stop_id	**destination_stop_id**	**total_count**
stop1	stop2	523
stop1	stop4	661

stop1	stop5	345
stop1	stop6	122
stop1	stop13	8213
stop1	stop22	3244
stop1	…all destinations…	…

```
                        ]
                }
        ]
}
```

EXAMPLE RESPONSE IN THE SWAGGER FILE FOR

- API endpoint: /api/v1/compute/passengers/stop-to-stop/average

Example 3 [multiple-days & multiple origin-destination & group by time]: I want to know the total number of passengers who started their journeys from: 'Zurich-Altstetten' (= stop1), and ‘Zurich HB’ (=stop3); with destination to 'Zurich-Oerlikon' (= stop2), ‘Bern HB’ (=stop4), ‘Basel HB Busstop’ (=stop5), and ‘Genève’ (stop=6) in between January 26 and January 29, group by days.

Input:

```
{
        “default_country”: “CH”,
        "time_interval":{
                “start_timestamp”: “2017-01-26 00:00:00”,
                “end_timestamp”: “2017-01-29 23:59:59”
        },
        “origin_stops”: [
                {
                        “stop_id”: “stop1”
                },
                {
                        “stop_id”: “stop3”
                }
        ],
        “destination_stops”: [
                {
                        “stop_id”: “stop2”
                },
                {
                        “stop_id”: “stop4”
                },
                {
                        “stop_id”: “stop5”
                },
                {
                     “stop_id”: “stop6”
                }

        ],
        “average_by”: “PER_DAY_OF_WEEK”
}
```

Output:

```
{
        “metadata”:
        {
                “api_version”: “v1”
        },
        “request”: {
                “default_country”: “CH”,
                "time_interval":{
                        “start_timestamp”: “2017-01-26 00:00:00”,
                        “end_timestamp”: “2017-01-29 23:59:59”
                },
                 “origin_stops”: [
                        {
                                “stop_id”: “stop1”
                        },
                        {
                                “stop_id”: “stop3”
                        }
                ],
                “destination_stops”: [
                        {
                                “stop_id”: “stop2”
                        },
                        {
                                “stop_id”: “stop4”
                        },
                        {
                                “stop_id”: “stop5”
                        },
                        {
                                “stop_id”: “stop6”
                        }

                ],
                “average_by”: “PER_DAY_OF_WEEK”
        },
        “journeys”: [
                {
                        "time_interval":{
```

“start_timestamp”: “2017-01-26 00:00:00”,
“end_timestamp”: “2017-01-26 23:59:59”
},
“od_matrix”: [

origin_stop_id	**destination_stop_id**	**total_count**
stop1	stop2	523
stop1	stop4	661
stop1	stop5	345
stop1	stop6	122
stop3	stop2	10234
stop3	stop4	33124
stop3	stop5	23432
stop3	stop6	44412

]
},
{
"time_interval":{
“start_timestamp”: “2017-01-27 00:00:00”,
“end_timestamp”: “2017-01-27 23:59:59”
},
“od_matrix”: [

origin_stop_id	**destination_stop_id**	**total_count**
stop1	stop2	323
stop1	stop4	61
stop1	stop5	545
stop1	stop6	162
stop3	stop2	12234
stop3	stop4	53124
stop3	stop5	3432
stop3	stop6	43472

]
},
{
"time_interval":{
“start_timestamp”: “2017-01-28 00:00:00”,
“end_timestamp”: “2017-01-28 23:59:59”
},
“od_matrix”: [

origin_stop_id	**destination_stop_id**	**total_count**
stop1	stop2	723
stop1	stop4	861
stop1	stop5	945
stop1	stop6	622
stop3	stop2	40234
stop3	stop4	34124
stop3	stop5	21432
stop3	stop6	87412

]
},

```
                {
                        "time_interval":{
                                "start_timestamp": "2017-01-29 00:00:00",
                                "end_timestamp": "2017-01-29 23:59:59"
                        },
                        "od_matrix": [
```

origin_stop_id	destination_stop_id	total_count
stop1	stop2	543
stop1	stop4	612
stop1	stop5	356
stop1	stop6	127
stop3	stop2	17234
stop3	stop4	398124
stop3	stop5	26432
stop3	stop6	47992

```
                        ]
                }

        ]
}
```

Example of endpoint /api/v1/compute/passengers/stop-to-stop/average/

EXAMPLE RESPONSE IN THE SWAGGER FILE FOR

- API endpoint: /api/v1/compute/passengers/stop-to-stop/average

Example 4 [multiple-days & multiple origin-destination & aggregate as stations & average count & search by and output stop names]: I want to know the average number of passenger per day in a week who started their journeys from the stations (i.e. all the outgoing journeys of all the platforms from this parent-stop-station): 'Zurich HB' (= stop3001), and 'Bahnhofplatz/HB' (=stop3002); with destination to the stations 'Bern HB' (=stop3005), 'Basel HB' (=stop3008), and 'Genève' (stop=3016) in between January 1 and January 31.

<u>*Input:*</u>

```
{
        "default_country": "CH",
        "time_interval":
        {
                "start_timestamp": "2017-01-01 00:00:00",
                "end_timestamp": "2017-01-31 23:59:59",
        },
        "origin_stops": [
                {
                         "stop_name": "Zurich HB"
                },
                {
                        “stop_name”: “Bahnhofplatz/HB”
                }
        ],
        “destination_stops”: [
                {
                        “stop_name”: “Bern HB”
                },
                {
                        “stop_ name”: “Basel HB”
                },
                {
                        “stop_ name”: “Genève”
                }
        ],
        “aggregate_all_origin_stops_as_stations”: “TRUE”,
        “aggregate_all_destination_stops_as_stations”: “TRUE”,
        “od_matrix_output”: “STOP_NAME”,
        “average_by”: “PER_DAY_OF_WEEK”

}
```

Output:

```
{
        “metadata”:
        {
                “api_version”: “v1”
        },
        “request”: {
                “default_country”: “CH”,
                “time_interval”:
                {
                        “start_timestamp”: “2017-01-01 00:00:00”,
                        “end_timestamp”: “2017-01-31 23:59:59”,
                },
                “origin_stops”: [
                        {
                                “stop_name”: “Zurich HB”
                        },
                        {
                                “stop_name”: “Bahnhofplatz/HB”
                        }
                ],
                “destination_stops”: [
                        {
                                “stop_name”: “Bern HB”
                        },
                        {
                                “stop_ name”: “Basel HB”
                        },
                        {
                                “stop_ name”: “Genève”
                        }
                ],
                “aggregate_all_origin_stops_as_stations”: “TRUE”,
                “aggregate_all_destination_stops_as_stations”: “TRUE”,
                “od_matrix_output”: “STOP_NAME”,
                “average_by”: “PER_DAY_OF_WEEK”
        },
        “journeys”: [
                {
                        “time_interval”:
                        {
                                “start_timestamp”: “2017-01-01 00:00:00”,
                                “end_timestamp”: “2017-01-31T23:59:59”
                        },
                         “average_interval_value”: “1”, (/*mondays*/)
                        “od_matrix”: [
```

origin_stop_name	**destination_stop_name**	**average**
Zurich HB	*Bern HB*	523123
Zurich HB	*Basel HB*	661134
Zurich HB	*Genève*	345324
Bahnhofplatz/HB	*Bern HB*	521326
Bahnhofplatz/HB	*Basel HB*	610234
Bahnhofplatz/HB	*Genève*	373124

```
                ]
        },
        {
                “time_interval”:
                {
                        “start_timestamp”: “2017-01-01 00:00:00”,
                        “end_timestamp”: “2017-01-31 23:59:59”
                },
                 “average_interval_value”: “2”,
                “od_matrix”: [
```

origin_stop_name	**destination_stop_name**	**average**
Zurich HB	*Bern HB*	313443
Zurich HB	*Basel HB*	53544
Zurich HB	*Genève*	34567
Bahnhofplatz/HB	*Bern HB*	975444
Bahnhofplatz/HB	*Basel HB*	782111
Bahnhofplatz/HB	*Genève*	568999

```
                ]

        },
        {
                …average_interval_values from 3-wednesday to 6-saturday…
        },
        {
                “time_interval”:
                {
                        “start_timestamp”: “2017-01-01 00:00:00”,
                        “end_timestamp”: “2017-01-31 23:59:59”
                },
                “average_interval_value”: “7”,
                “od_matrix”: [
```

origin_stop_name	**destination_stop_name**	**average**
Zurich HB	*Bern HB*	576123
Zurich HB	*Basel HB*	612134
Zurich HB	*Genève*	985324
Bahnhofplatz/HB	*Bern HB*	18326
Bahnhofplatz/HB	*Basel HB*	368234
Bahnhofplatz/HB	*Genève*	951124

```
                        ]

                }

        ]
}
```

Example 5 [multiple-days & multiple origin-destination & aggregate as stations & average count & search by and output stop names]: I want to know the average number of passengers per day in the total time interval who started their journeys from the stations (i.e. all the outgoing journeys of all the platforms from this parent-stop-station): 'Zurich HB' (= stop3001), and ‘Bahnhofplatz/HB’ (=stop3002); with destination to the stations ‘Bern HB’ (=stop3005), ‘Basel HB’ (=stop3008), and ‘Genève’ (stop=3016) in between January 1 and January 31.

Input:

```
{
        “default_country”: “CH”,
        "time_interval":
        {
                “start_timestamp”: “2017-01-01 00:00:00”,
                “end_timestamp”: “2017-01-31 23:59:59”
        },
        “origin_stops”: [
                {
                        “stop_name”: “Zurich HB”
```

```
		},
		{
			“stop_name”: “Bahnhofplatz/HB”
		}
	],
	“destination_stops”: [
		{
			“stop_name”: “Bern HB”
		},
		{
			“stop_ name”: “Basel HB”
		},
		{
			“stop_ name”: “Genève”
		}
	],
	 “aggregate_all_origin_stops_as_stations”: “TRUE”,
	“aggregate_all_destination_stops_as_stations”: “TRUE”,
	“average_by”: “TOTAL_DAYS_OF_TIME_INTERVAL”

}
```

Output:

```
{
	“metadata”:
	{
		“api_version”: “v1”
	},
	 “request”: {
		“default_country”: “CH”,
		"time_interval":
		{
			“start_timestamp”: “2017-01-01 00:00:00”,
			“end_timestamp”: “2017-01-31 23:59:59”
		},
		 “origin_stops”: [
			{
				“stop_name”: “Zurich HB”
			},
			{
				“stop_name”: “Bahnhofplatz/HB”
			}
		],
		“destination_stops”: [
			{
				“stop_name”: “Bern HB”
			},
                        {
                                “stop_ name”: “Basel HB”
                        },
                        {
                                “stop_ name”: “Genève”
                        }
                ],
                 “aggregate_all_origin_stops_as_stations”: “TRUE”,
                “aggregate_all_destination_stops_as_stations”: “TRUE”,
                “average_by”: “TOTAL_DAYS_OF_TIME_INTERVAL”
        },
        “journeys”: [
                {
                        "time_interval":
                        {
                                “start_timestamp”: “2017-01-01 00:00:00”,
                                “end_timestamp”: “2017-01-31 23:59:59”
                        },
                         “od_matrix”: [
```

origin_stop_name	**destination_stop_name**	**average** *(total_count / 31)*
Zurich HB	*Bern HB*	83123
Zurich HB	*Basel HB*	91134
Zurich HB	*Genève*	39324
Bahnhofplatz/HB	*Bern HB*	9326
Bahnhofplatz/HB	*Basel HB*	2234
Bahnhofplatz/HB	*Genève*	3324

```
                        ]
                }
        ]
}
```

Example 6 [multiple-days & multiple origin-destination & aggregate as stations & average count & search by and output stop names & average by time interval]: I want to know the average number of passengers per day who started their journeys from the stations (i.e. all the outgoing journeys of all the platforms from this parent-stop-station): 'Zurich HB' (= stop3001), and ‘Bahnhofplatz/HB’ (=stop3002); with destination to the stations ‘Bern HB’ (=stop3005), ‘Basel HB’ (=stop3008), and ‘Genève’ (stop=3016) in between January 1 and February 28 .

<u>Input:</u>

```
{
        “default_country”: “CH”,
        "time_interval":
```

```
	{
		“start_timestamp”: “2017-01-01 00:00:00”,
		“end_timestamp”: “2017-02-28 23:59:59”
	},
	“origin_stops”: [
		{
			“stop_name”: “Zurich HB”
		},
		{
			“stop_name”: “Bahnhofplatz/HB”
		}
	],
	“destination_stops”: [
		{
			“stop_name”: “Bern HB”
		},
		{
			“stop_ name”: “Basel HB”
		},
		{
			“stop_ name”: “Genève”
		}
	],
	“aggregate_all_origin_stops_as_stations”: “TRUE”,
	“aggregate_all_destination_stops_as_stations”: “TRUE”,
	“average_by”: “TOTAL_DAYS_OF_TIME_INTERVAL”

}
```

Output:

```
{
	“metadata”:
	{
		“api_version”: “v1”
	},

	“request”: {
		“default_country”: “CH”,
		"time_interval":
		{
			“start_timestamp”: “2017-01-01 00:00:00”,
			“end_timestamp”: “2017-02-28 23:59:59”
		},
		 “origin_stops”: [
			{
				“stop_name”: “Zurich HB”
			},
                        {
                                    “stop_name”: “Bahnhofplatz/HB”
                        }
            ],
            “destination_stops”: [
                        {
                                    “stop_name”: “Bern HB”
                        },
                        {
                                    “stop_ name”: “Basel HB”
                        },
                        {
                                    “stop_ name”: “Genève”
                        }
            ],
            “aggregate_all_origin_stops_as_stations”: “TRUE”,
            “aggregate_all_destination_stops_as_stations”: “TRUE”,
            “average_by”: “TOTAL_DAYS_OF_TIME_INTERVAL”
    },
    “journeys”: [
            {
                        "time_interval":
                        {
                                   “start_timestamp”: “2017-01-01 00:00:00”,
                                   “end_timestamp”: “2017-02-28 23:59:59”
                        },
                        “average_interval_value”: “59” (total # of days between 01-01 and
                        02-28)
                        “od_matrix”: [
```

origin_stop_name	**destination_stop_name**	**average** *(total count / 59)*
Zurich HB	*Bern HB*	3256
Zurich HB	*Basel HB*	53479
Zurich HB	*Genève*	31741
Bahnhofplatz/HB	*Bern HB*	1231
Bahnhofplatz/HB	*Basel HB*	4122
Bahnhofplatz/HB	*Genève*	4424

```
                        ]
            }
    ]
}
```

4.2.2. API endpoint: /api/v1/compute/passengers/stop-to-stop/average

See Example 4.

4.2.3. API endpoint: /api/v1/compute/passengers/transfer/trip-to-trip/count

Text only for /api/v1/compute/passengers/transfer/trip-to-trip/waiting-time

Description:
Given a list of incoming trips, this endpoint calculates the total number of passengers that arrived at a specific "arrival time", transferred at a "transfer_stop", and continue their journey at a specific "departure time" to a different list of outgoing trips. When no result is found, the OD matrix entry will be non-existent and will be omitted in the return output.

This computation counts all the passengers that transferred from one specific vehicle (incoming trip, ex. "trip_id= 283957") arriving at a specific time (ex. "2017-01-27 08:05:00") on a specific "transfer stop" (ex. "Bern HB"), and continue their journeys with another specific vehicle (outgoing trip, ex. "trip_id= 559") departing at a specific time (ex. "2017-01-27 08:17:00") from the same "transfer stop".

Computations outputs:
- TOTAL COUNT
  - Endpoint: **/api/v1/compute/passengers/transfer/trip-to-trip/count**
  - Inputs: incoming/outgoing trips
  - Description: returns the total count of passengers that transfer to a new outgoing trip.
- WAITING TIME
  - Endpoint: **/api/v1/compute/passengers/transfer/trip-to-trip/waiting-time**
  - Description: shows the waiting time (difference in between trip_arrival_time of the incoming trip, and the trip_departure_time of the outgoing trip) of passengers in between trips.

HTTP Verb: POST

Input (send JSON request):

default_country: (required) CH (Switzerland), SE (Sweden), NL (Netherlands),

incoming_trips: (required) Array of
   trip_id: STRING (unique in DB),
   trip_arrival_time: DATE (required, arrival time of this trip at the "transfer_stop"),
   country: (optional, overrides "default_country" only for this element) CH (Switzerland), SE (Sweden), NL (Netherlands),

transfer_stops: (required) Array of
   stop_id: STRING (required if "stop_name" is empty),
   stop_name: STRING (required if "stop_id" is empty),

country: (optional, overrides "default_country" only for this element) CH (Switzerland), SE (Sweden), NL (Netherlands),
aggregate_as_station: Boolean (Default false. If true, this stop is treated as a station and the calculation will aggregate all the children stop of this station. If false, this stop is treated as a single atomic stop).

outgoing_trips: (required) Array of
trip_id: STRING (unique in DB),
trip_departure_time: DATE (required, departure time of this trip at the "transfer_stop"),
country: (optional, overrides "default_country" only for this element) CH (Switzerland), SE (Sweden), NL (Netherlands),

Output (expect JSON response):

request: SAME AS ORIGINAL REQUEST,

metadata:
api_version: v1

transfers: Array of (one per transfer_stop)

transfer_stop_id: STRING (shown if "stop_name" is empty),
transfer_stop_name: STRING (shown if "stop_id" is empty),
transfer_stop_country (only appears if transfer_stop[country] is not empty): CH (Switzerland), SE (Sweden), NL (Netherlands),

od_matrix: Array of (one per incoming_trip to outgoing_trip combination)
incoming_trip_country (only appears if incoming_trips[country] is not empty): CH (Switzerland), SE (Sweden), NL (Netherlands),
incoming_trip_id: STRING,
incoming_trip_arrival_time: DATE,
outgoing_trip_country (only appears if outgoing_trips[country] is not empty): CH (Switzerland), SE (Sweden), NL (Netherlands),
outgoing_trip _id: STRING,
outgoing_trip_departure_time: DATE,
total_count: Integer
waiting_time: Integer (in seconds)

Example of endpoint:
/api/v1/compute/passengers/transfer/trip-to-trip/count

EXAMPLE RESPONSE IN THE SWAGGER FILE FOR

- API endpoint: /api/v1/compute/passengers/transfer/trip-to-trip/count

Example 1 [passenger count]. I want to know the total number of passengers that transferred at Bern HB (= stop_name) or “Bahnhof Olten” (= stop_name), arriving/using the “111111” (=trip_id) at 2017-01-26 18:15:00 or “222222” (=trip_id) at 2017-01-26 18:21:00; and continue their journey with either “333333” (=trip_id) at 2017-01-26 18:35:00, “444444” (=trip_id) at 2017-01-26 18:38:00, “555555” (=trip_id) at 2017-01-26 18:33:00 or “666666” (=trip_id) at 2017-01-26 18:35:00.

Input:

```
{
        “default_country”: “CH”,
         “incoming_trips”: [
                {
                        “trip_id”: “111111”,
                        “trip_arrival_time”: “2017-01-26 18:15:00”
                },
                {
                        “trip_id”: “222222”,
                        “trip_arrival_time”: “2017-01-26 18:21:00”
                }
        ],
        “transfer_stops”:
        [
                {
                        “stop_name”: “Bern HB”,
                        “stop_name”: “Bahnhof Olten”
                }
        ],
        “outgoing_trips”: [
                {
                         “trip_id”: “333333”,
                        “trip_departure_time”: “2017-01-26 18:35:00”

                },
                {
                         “trip_id”: “444444”,
                        “trip_departure_time”: “2017-01-26 18:38:00”
                },
                {
                         “trip_id”: “555555”,
                        “trip_departure_time”: “2017-01-26 18:33:00”
                },
                {
                         “trip_id”: “666666”,
                        “trip_departure_time”: “2017-01-26 18:35:00”
```

```
		}
	]
}
```

<u>*Output:*</u>

```
{

	“metadata”:
	{
		“api_version”: “v1”
	},

	“request”: {
		“default_country”: “CH”,
		 “incoming_trips”: [
			{
				“trip_id”: “111111”,
				“trip_arrival_time”: “2017-01-26 18:15:00”
			},
			{
				“trip_id”: “222222”,
				“trip_arrival_time”: “2017-01-26 18:21:00”
			}
		],
		“transfer_stops”:
		[
			{
				“stop_name”: “Bern HB”,
				“stop_name”: “Bahnhof Olten”
			}
		],
		“outgoing_trips”: [
			{
				 “trip_id”: “333333”,
				“trip_departure_time”: “2017-01-26 18:35:00”

			},
			{
				 “trip_id”: “444444”,
				“trip_departure_time”: “2017-01-26 18:38:00”
			},
			{
				 “trip_id”: “555555”,
				“trip_departure_time”: “2017-01-26 18:33:00”
			},
			{
				 “trip_id”: “666666”,
				“trip_departure_time”: “2017-01-26 18:35:00”
```

```
                    }
            ]

    },

    “transfers”:
    [
            {
                    “transfer_stop_name”: “Bern HB”,
                     “od_matrix”:
                    [
```

incoming_trip_id	incoming_trip_arrival_time	outgoing_trip_id	outgoing_trip_departure_time	total_count
222222	2017-01-26 18:21:00	444444	2017-01-26 18:38:00	15
222222	2017-01-26 18:21:00	555555	2017-01-26 18:33:00	4
222222	2017-01-26 18:21:00	666666	2017-01-26 18:35:00	9

```
                    ]
            },
            {
                    “transfer_stop_name”: “Bahnhof Olten”,
                     “od_matrix”:
                    [
```

incoming_trip_id	incoming_trip_arrival_time	outgoing_trip_id	outgoing_trip_departure_time	total_count
111111	2017-01-26 18:15:00	333333	2017-01-26 18:35:00	4

```
                    ]
            },


    ]
}
```

Example of endpoint:
/api/v1/compute/passengers/transfer/trip-to-trip/waiting-time

EXAMPLE RESPONSE IN THE SWAGGER FILE FOR

- API endpoint: /api/v1/compute/passengers/transfer/trip-to-trip/waiting-time

Example 2 [waiting time]. I want to know the waiting time of passengers that transferred at Bern HB (= stop_name) or “Bahnhof Olten” (= stop_name), arriving/using the “111111” (=trip_id) at 2017-01-26 18:15:00 or “222222” (=trip_id) at 2017-01-26 18:21:00; and continue their journey with either “333333” (=trip_id) at 2017-01-26 18:35:00, “444444”

(=trip_id) at 2017-01-26 18:38:00, “555555” (=trip_id) at 2017-01-26 18:33:00 or “666666” (=trip_id) at 2017-01-26 18:35:00.

Input:

```
{
        “default_country”: “CH”,
         “incoming_trips”: [
                {
                        “trip_id”: “111111”,
                        “trip_arrival_time”: “2017-01-26 18:15:00”
                },
                {
                        “trip_id”: “222222”,
                        “trip_arrival_time”: “2017-01-26 18:21:00”
                }
        ],
        “transfer_stops”:
        [
                {
                        “stop_name”: “Bern HB”,
                        “stop_name”: “Bahnhof Olten”
                }
        ],
        “outgoing_trips”: [
                {
                         “trip_id”: “333333”,
                        “trip_departure_time”: “2017-01-26 18:35:00”

                },
                {
                         “trip_id”: “444444”,
                        “trip_departure_time”: “2017-01-26 18:38:00”
                },
                {
                         “trip_id”: “555555”,
                        “trip_departure_time”: “2017-01-26 18:33:00”
                },
                {
                         “trip_id”: “666666”,
                        “trip_departure_time”: “2017-01-26 18:35:00”
                }
        ]
}
```

Output:

```
{
```

```
“metadata”:
{
        “api_version”: “v1”
},

“request”: {
        “default_country”: “CH”,
         “incoming_trips”: [
                {
                        “trip_id”: “111111”,
                        “trip_arrival_time”: “2017-01-26 18:15:00”
                },
                {
                        “trip_id”: “222222”,
                        “trip_arrival_time”: “2017-01-26 18:21:00”
                }
        ],
        “transfer_stops”:
        [
                {
                        “stop_name”: “Bern HB”,
                        “stop_name”: “Bahnhof Olten”
                }
        ],
        “outgoing_trips”: [
                {
                         “trip_id”: “333333”,
                        “trip_departure_time”: “2017-01-26 18:35:00”

                },
                {
                         “trip_id”: “444444”,
                        “trip_departure_time”: “2017-01-26 18:38:00”
                },
                {
                         “trip_id”: “555555”,
                        “trip_departure_time”: “2017-01-26 18:33:00”
                },
                {
                         “trip_id”: “666666”,
                        “trip_departure_time”: “2017-01-26 18:35:00”
                }
        ]

},

“transfers”:
[
        {
```

“transfer_stop_name”: “Bern HB”,
“od_matrix”:
[

incoming_trip_id	incoming_trip_arrival_time	outgoing_trip_id	outgoing_trip_departure_time	waiting_time
222222	2017-01-26 18:21:00	444444	2017-01-26 18:38:00	2434
222222	2017-01-26 18:21:00	555555	2017-01-26 18:33:00	4432
222222	2017-01-26 18:21:00	666666	2017-01-26 18:35:00	1314

]
},
{
“transfer_stop_name”: “Bahnhof Olten”,
“od_matrix”:
[

incoming_trip_id	incoming_trip_arrival_time	outgoing_trip_id	outgoing_trip_departure_time	waiting_time
111111	2017-01-26 18:15:00	444444	2017-01-26 18:38:00	2334

]
},

]
}

4.2.4. API endpoint: /api/v1/compute/passengers/transfer/trip-to-trip/waiting-time

Check API endpoint: /api/v1/compute/passengers/transfer/trip-to-trip/count

See Example 2.

4.2.5. API endpoint: /api/v1/compute/passengers/transfer/route-to-route/count

Text only for /api/v1/compute/passengers/transfer/route-to-route/average

Text only for /api/v1/compute/passengers/transfer/route-to-route/waiting-time/distribution

Description:
Given a list of incoming routes, this endpoint calculates the total number (or average) of passengers that transfer at a "transfer_stop", and continue their journey to a different list of outgoing routes. When an incoming route doesn't stop at the "transfer_stop" (or it doesn't transfer to the outgoing route), the OD matrix entry will be omitted in the response.

This computation counts all the passengers that enter the vehicle of a specific route (for example "route_id=7077") in any possible previous stop than the "transfer_stop" withtin the same route (e.g., if the transfer is "transfer_stop_name=Bern", which is the stop number 4 of the route 7077, then get all the passengers that started their journey using the same route 7077 at any of the stops with sequence number 1, 2, and 3), to then later continue their journey to another route (e.g., "route_id=3124").

Computations outputs:

- TOTAL COUNT
  - Endpoint: **/api/v1/compute/passengers/transfer/route-to-route/count**
  - Description: returns the total number of passengers that transfer from the incoming to the outgoing route.
- AVERAGE
  - Endpoint: **api/v1/compute/passengers/transfer/route-to-route/average**
  - Description: returns the average number of passengers that transfer from the incoming to the outgoing route, according to the "average_by" filter.
- WAITING TIME DISTRIBUTION
  - Endpoint: **/api/v1/compute/passengers/transfer/route-to-route/waiting-time/distribution**
  - Description: returns the distribution of the waiting time (difference in between arrival_time of the incoming trip and the departure_time of the outgoing trip of the given incoming and outgoing routes) of passengers. The (cumulative) distribution is provided at a list of percentiles for the given percentile_increments (default is 5%, that means values of the distribution at the 5%, 10%, 15%, ..., 95%). The percentile increment can be specified.

HTTP Verb: POST

Input (send JSON request):

default_country: (required) CH (Switzerland), SE (Sweden), NL (Netherlands),

time_interval:

start_timestamp: Date (required),
end_timestamp: Date (required),

incoming_routes: Array of
route_id: STRING (unique, required if route_short_name is empty),
route_short_name: STRING (NOT unique, required if route_id is empty),
country: (optional, overrides "default_country" only for this element) CH (Switzerland), SE (Sweden), NL (Netherlands),

transfer_stops: Array of
stop_id: STRING (required),
country: (optional, overrides "default_country" only for this element) CH (Switzerland), SE (Sweden), NL (Netherlands),

outgoing_routes: Array of
route_id: STRING (unique in DB, require if route_short_name is empty),
route_short_name: STRING (NOT unique in DB, required if route_short_id is empty),
country: (optional, overrides "default_country" only for this element) CH (Switzerland), SE (Sweden), NL (Netherlands),

average_by (required only for endpoint …route-to-route/average):
TOTAL_HOURS_OF_TIME_INTERVAL: total count of passenger journeys divided by the number of hours in between the end_timestamp and start_timestamp;
TOTAL_DAYS_OF_TIME_INTERVAL, Default, total count of passenger journeys divided by the number of days in between the end_timestamp and start_timestamp;

distribution_percentile_interval: Integer (required only for endpoint: "…/distribution", input the percentile interval discretization value in between 1 and 100 for the desired output computation.)

Output (expect JSON response):

request: SAME AS ORIGINAL REQUEST,

metadata:
api_version: v1

transfers: Array of (If specified, one per transfer_stop, or average_by value, in between the total time_interval; if none of the above is specified, one for the complete time_interval)

transfer_stop_id: STRING,

transfer_stop_country (only appears if transfer_stop[country] is not empty): CH (Switzerland), SE (Sweden), NL (Netherlands),

time_interval: (required)
  start_timestamp: Date (required),
  end_timestamp: Date (required),

average_interval_value: STRING (only for endpoint "/api/v1/compute/passengers/stop-to-stop/average")

- For the type of TOTAL_X_PER_TIME_INTERVAL:
  - "17" [if TOTAL_HOURS_PER_TIME_INTERVAL, in between "2016-10-03 06:00:00" and "2016-10-03 23:00:00")
  - "61" [if TOTAL_DAYS_PER_TIME_INTERVAL, in between 01-nov to 31-dec])

od_matrix: Array of (one per incoming_route to outgoing_route combination)
  incoming_route_id: STRING,
  incoming_route _country (only appears if incoming_trips[country] is not empty): CH (Switzerland), SE (Sweden), NL (Netherlands),
  outgoing_route _id: STRING,
  outgoing_route_country (only appears if outgoing_trips[country] is not empty): CH (Switzerland), SE (Sweden), NL (Netherlands),
  total_count: Integer (only for endpoint: "…/count")
  average: Double (only for endpoints "…/average")

distribution_matrix (Array, only appears for "…/distribution" endpoint)
  percentile: INTEGER (ex. "10", means the 10th percentile, "20", means 20th percentile, etc.),
  waiting_time: INTEGER (in minutes)

Example of endpoint:
/api/v1/compute/passengers/transfer/route-to-route/count

EXAMPLE RESPONSE IN THE SWAGGER FILE FOR

- API endpoint: /api/v1/compute/passengers/transfer/route-to-route/count

Example 1 [Transfer passenger count]. I want to know the total number of passengers that started their journey with the routes "IC70112" (=route_id) or "IC70125" (=route_id), transferred at the stations "Bern HB" (=stop_name) or "Bahnhof Olten"(=stop_name), and continue their journey with any of the routes "IR3243" (=route_id), "S5432" (=route_id), "S3112" (=route_id) or "Bus20_trip20" (=route_id); in between the 26 and 29 of January.

Input:

```
{
        “default_country”: “CH”,
        "time_interval":{
                “start_timestamp”: “2017-01-26 00:00:00”,
                “end_timestamp”: “2017-01-29 23:59:59”
        },
        “incoming_routes”: [
                {
                        “route_id”: “IC70112”
                },
                {
                        “route_id”: “IC70125”
                }
        ],
        “transfer_stops”:
        [
                {
                        “stop_name”: “Bern HB”
                },
                {
                        “stop_name”: “Bahnhof Olten”
                }
        ],
        “outgoing_routes”: [
                {
                         “route_id”: “IR3243”
                },
                {
                         “route_id”: “S5432”
                },
                {
                         “route_id”: “S3112”
                },
                {
                         “route_id”: “Bus20_trip20”
                }
        ]
}
```

Output:

```
{
        “metadata”:
        {
                “api_version”: “v1”
        },
        “request”: {
                “default_country”: “CH”,
                "time_interval":{
                        “start_timestamp”: “2017-01-26 00:00:00”,
                        “end_timestamp”: “2017-01-29 23:59:59”
                },
                “incoming_routes”: [
                        {
                                “route_id”: “IC70112”
                        },
                        {
                                “route_id”: “IC70125”
                        }
                ],
                “transfer_stops”:
                [
                        {
                                “stop_name”: “Bern HB”
                        },
                        {
                                “stop_name”: “Bahnhof Olten”
                        }
                ],
                 “outgoing_routes”: [
                        {
                                 “route_id”: “IR3243”
                        },
                        {
                                 “route_id”: “S5432”
                        },
                        {
                                 “route_id”: “S3112”
                        },
                        {
                                 “route_id”: “Bus20_trip20”
                        }
                ]

        },

        “transfers”:
        [
                {
                        "time_interval":{
                                “start_timestamp”: “2017-01-26 00:00:00”,
                                “end_timestamp”: “2017-01-29 23:59:59”
                        },
                        “transfer_stop_name”: “Bern HB”,
                        “od_matrix”:
                        [
```

incoming_route_id	outgoing_route_id	total_count
IC70112	IR3243	4
IC70112	S5432	25
IC70125	IR3243	5
IC70125	S5432	15
IC70125	S3112	4

```
                ]
            },
            {
                "time_interval":{
                    “start_timestamp”: “2017-01-26 00:00:00”,
                    “end_timestamp”: “2017-01-29 23:59:59”
                },
                 “transfer_stop_name”: “Bahnhof Olten”,
                “od_matrix”:
                [
```

incoming_route_id	outgoing_route_id	total_count
IC70112	IR3243	11
IC70112	S5432	43
IC70112	Bus20_trip20	75
IC70125	IR3243	24
IC70125	S5432	56
IC70125	S3112	89
IC70125	Bus20_trip20	12

```
                ]
            }

        ]
}
```

Example of Endpoint: **api/v1/compute/passengers/transfer/route-to-route/average**

EXAMPLE RESPONSE IN THE SWAGGER FILE FOR

- API endpoint: /api/v1/compute/passengers/transfer/route-to-route/average

Example 2 [Transfer passenger count average]. I want to know the daily average (taken over the total amount of days in between a time interval) number of passengers that started their journey with the routes “IC70112” (=route_id) or “IC70125” (=route_id), transferred at the stations “Bern HB” (=stop_name) or “Bahnhof Olten”(=stop_name), and continue their journey with either of the routes “IR3243” (=route_id), “S5432” (=route_id), “S3112” (=route_id) or “Bus20_trip20” (=route_id); in between the 26 and 29 of January.

Input:

```
{
    “default_country”: “CH”,
	"time_interval":{
		“start_timestamp”: “2017-01-26 00:00:00”,
		“end_timestamp”: “2017-01-29 23:59:59”
	},
	“incoming_routes”: [
		{
			“route_id”: “IC70112”
		},
		{
			“route_id”: “IC70125”
		}
	],
	“transfer_stops”:
	[
		{
			“stop_name”: “Bern HB”
		},
		{
			“stop_name”: “Bahnhof Olten”
		}
	],
	“outgoing_routes”: [
		{
			 “route_id”: “IR3243”
		},
		{
			 “route_id”: “S5432”
		},
		{
			 “route_id”: “S3112”
		},
		{
			 “route_id”: “Bus20_trip20”
		}
	],
	“average_by”: “TOTAL_DAYS_OF_TIME_INTERVAL”
}
```

Output:

```
{

	“metadata”:
	{
		“api_version”: “v1”
	},

	“request”: {
		“default_country”: “CH”,
        "time_interval":{
            “start_timestamp”: “2017-01-26 00:00:00”,
            “end_timestamp”: “2017-01-29 23:59:59”
        },
        “incoming_routes”: [
            {
                “route_id”: “IC70112”
            },
            {
                “route_id”: “IC70125”
            }
        ],
        “transfer_stops”:
        [
            {
                “stop_name”: “Bern HB”
            },
            {
                “stop_name”: “Bahnhof Olten”
            }
        ],
         “outgoing_routes”: [
            {
                 “route_id”: “IR3243”
            },
            {
                 “route_id”: “S5432”
            },
            {
                 “route_id”: “S3112”
            },
            {
                 “route_id”: “Bus20_trip20”
            }
        ] ,
        “average_by”: “TOTAL_DAYS_OF_TIME_INTERVAL”
    },

    “transfers”:
    [
        {
            "time_interval":{
                “start_timestamp”: “2017-01-26 00:00:00”,
                “end_timestamp”: “2017-01-29 23:59:59”
            },
             “transfer_stop_name”: “Bern HB”,
            “average_interval_value”: “4”,
            “od_matrix”:
            [
```

incoming_route_id	**outgoing_route_id**	**average**

IC70112	IR3243	3
IC70112	S5432	2
IC70125	IR3243	5
IC70125	S5432	15
IC70125	S3112	4

```
                ]
            },
            {
                "time_interval":{
                    “start_timestamp”: “2017-01-26 00:00:00”,
                    “end_timestamp”: “2017-01-29 23:59:59”
                },
                 “transfer_stop_name”: “Bahnhof Olten”,
                “average_interval_value”: “4”,
                “od_matrix”:
                [
```

incoming_route_id	outgoing_route_id	average
IC70112	IR3243	31
IC70112	S5432	22
IC70112	Bus20_trip20	55
IC70125	IR3243	12
IC70125	S5432	17
IC70125	S3112	33
IC70125	Bus20_trip20	11

```
                ]
            }

        ]
}
```

Example of endpoint:
/api/v1/compute/passengers/transfer/route-to-route/waiting-time/distribution

EXAMPLE RESPONSE IN THE SWAGGER FILE FOR

- API endpoint: /api/v1/compute/passengers/transfer/route-to-route/waiting-time/distribution

Example 3 [waiting time distribution]. I want to know the waiting time distribution (discretized in 25% percentile increments) of passengers that started their journey with the routes “IC70112” (=route_id) or “IC70125” (=route_id), transferred at the stations “Bern HB” (=stop_name) or “Bahnhof Olten”(=stop_name), and continue their journey with either of the routes “IR3243” (=route_id), “S5432” (=route_id), “S3112” (=route_id) or “Bus20_trip20” (=route_id); in between the 26 and 29 of January.

Input:

```
{
	“default_country”: “CH”,
	"time_interval":{
		“start_timestamp”: “2017-01-26 00:00:00”,
		“end_timestamp”: “2017-01-29 23:59:59”
	},
	“incoming_routes”: [
		{
			“route_id”: “IC70112”
		},
		{
			“route_id”: “IC70125”
		}
	],
	“transfer_stops”:
	[
		{
			“stop_name”: “Bern HB”
		},
		{
			“stop_name”: “Bahnhof Olten”
		}
	],
	“outgoing_routes”: [
		{
			 “route_id”: “IR3243”
		},
		{
			 “route_id”: “S5432”
		},
		{
			 “route_id”: “S3112”
		},
		{
			 “route_id”: “Bus20_trip20”
		}
	],
	“distribution_percentile_interval”: 25

}
```

Output:

```
{
	“metadata”:
	{
		“api_version”: “v1”
```

```
},

"request": {
        "default_country": "CH",
        "time_interval":{
                "start_timestamp": "2017-01-26 00:00:00",
                "end_timestamp": "2017-01-29 23:59:59"
        },
        "incoming_routes": [
                {
                        "route_id": "IC70112"
                },
                {
                        "route_id": "IC70125"
                }
        ],
        "transfer_stops":
        [
                {
                        "stop_name": "Bern HB",
                        "aggregate_as_station": true
                },
                {
                        "stop_name": "Bahnhof Olten",
                        "aggregate_as_station": true
                }
        ],
         "outgoing_routes": [
                {
                         "route_id": "IR3243"
                },
                {
                         "route_id": "S5432"
                },
                {
                         "route_id": "S3112"
                },
                {
                         "route_id": "Bus20_trip20"
                }
        ],
        "distribution_percentile_interval": 25
},

"transfers":
[
        {
                "time_interval":{
                        "start_timestamp": "2017-01-26 00:00:00",
                        "end_timestamp": "2017-01-29 23:59:59"
```

```
                    },
                     “transfer_stop_name”: “Bern HB”,
                     “distribution_matrix”:
                    [
```

percentile	waiting_time
0	2
25	2
50	5
75	7
100	12

```
                    ]
            },
            {
                    "time_interval":{
                            “start_timestamp”: “2017-01-26 00:00:00”,
                            “end_timestamp”: “2017-01-29 23:59:59”
                    },
                     “transfer_stop_name”: “Bahnhof Olten”,
                    “distribution_matrix”:
                    [
```

percentile	waiting_time
0	1
25	1
50	2
75	2
100	3

```
                    ]
            }
        ]
}
```

4.2.6. API endpoint: /api/v1/compute/passengers/transfer/route-to-route/average

Check API endpoint: /api/v1/compute/passengers/transfer/route-to-route/count

Example 2

4.2.7. API endpoint: /api/v1/compute/passengers/transfer/route-to-route/waiting-time/distribution

Check API endpoint: /api/v1/compute/passengers/transfer/route-to-route/count

Example 3

4.2.8. API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/count

Text only for /api/v1/compute/…/average

Text only for /api/v1/compute/…/distribution

Text only for /api/v1/compute/…/density

Description: Computations for the intra-station zones-to-zone. We define 3 types of computations:

1. **"Total count" or "average number" of passengers**: Get an OD-Matrix with the total (or the average) number of passengers (i.e., distinct passenger walks), who walked from one (or a list of multiple) origin intra-station zone(s) to another (or a list of multiple) destination intra-station zone(s), in a given time interval.
2. **Distribution**: Outputs the distribution of walking time [in seconds], of passengers who walked from one (or a list of multiple) origin intra-station zone(s) to another (or a list of multiple) destination intra-station zone(s), in a given time interval. Expect to receive the (cumulative) distribution as a list of percentiles at user-defined increments (default is 5%, that means values of the distribution at the 5%, 10%, 15%, ..., 95%). Other distribution computations: walking distance [in millimeters], or walking speed [in millimeters/second] of passengers,
3. **Density**: For a given cell discretization (a rectangular grid specified by its corners, length, and width), outputs the passenger density (= number of passengers per cell) in each grid cell. Other density computations: walking average speed (= average walking speed in each grid cell).

Computations outputs:

- PASSENGER_TOTAL_COUNT
  - Endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/count
  - Description: Returns the total count of passengers,
- PASSENGER_AVERAGE
  - Endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/average
  - Description: The average is calulated by the specification of the "average_by" attribute of the request;
- WALKING_DISTANCE_DISTRIBUTION:
  - Endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/walking-distance/distribution
  - Description: the (cumulative) distribution of walking distance (in mm) as a list of percentiles in default 5% increments (that is, values of the distribution at the 0%, 5%, 10%, 15%, ..., 95%, 100% quantiles),
- WALKING_TIME_DISTRIBUTION:
  - Endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/walking-time/distribution

    - Description: the (cumulative) distribution of walking time (in seconds) as a list of percentiles in default 5% increments (that is, values of the distribution at the 0%, 5%, 10%, 15%, ..., 95%, 100% quantiles),
- WALKING_SPEED_DISTRIBUTION:
    - Endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/walking-speed/distribution
    - Description: the (cumulative) distribution of walking speed (distance/time, in mm/s) as a list of percentiles in default 5% increments (that is, values of the distribution at the 0%, 5%, 10%, 15%, ..., 95%, 100% quantiles)
- PASSENGER_DENSITY
    - Endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/density
    - Description: Average number of passengers per in each grid cell at a given time interval,
- WALKING_SPEED_AVERAGE_DENSITY:
    - Endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/walking-speed/average/density
    - Description: Average walking speed of passengers in each grid cell in a given time interval.

Definitions:
- “Pre-defined zones”: Pre-defined intra-station zones are station-dependent. As an example, Lausanne Station has 25 different zones.
- “Grid cells”: Each station is divided in a 16m^2 (4m x 4m) grid and each of the inner cells represent an intra-station cell identified by its “cell_id” (unique value in the Graph DB). To identify a specific cell in a station, we use the format “A-B”, where “A” is the column number, increments from left-to-right in the x axis; and “B” is the row number, increments from top-to-down in the y axis, ex. 1-1, 2-1, 3-1,…).
- Coordinates “0,0” is the top-left corner in a map in each hub/station.
- Units:
    - Distance: millimeters
    - Time: seconds
    - Speed: millimeters / second

HTTP Verb: POST

Input (send JSON request):

default_country: (required) CH (Switzerland), SE (Sweden), NL (Netherlands),

default_parent_station: (required)

stop_id: (STRING, required only if stop_name is empty) - GTFS “stop_id” unique value within a country, for example “17382”,

stop_name: (STRING, required only if stop_id is empty) - GTFS “ stop_name” NOT unique value within a country, e.g., “Lausanne”

time_interval:
- ***start_timestamp***: Date (required),
- ***end_timestamp***: Date (required),

od_input_type: ZONE_LIST (match each origin zones from a specific origin list to each destination zones from the destination list, for all endpoints except “.../density”), GRID (match every cell inside the grid to all the other cells, only for endpoints “.../density”),

grid_details: (required only if “od_input_type=GRID”)
- ***grid_boundaries:*** (required)
  - ***corner***: (top-left corner)
    - ***x***: INTEGER (in mm)
    - ***y***: INTEGER (in mm)
  - ***length***: Integer (in mm, value in the x axis discretization. Pre-defined inner cells boundaries of the grid are 4000mm x 4000mm each, so this value will be rounded up to the closest 4000mm multiple, e.g., if “3000”, then “4000” will be used)
  - ***width***: Integer (in mm, value in the y axis. Pre-defined inner cells boundaries of the grid are 4000mm x 4000mm each, so this value will be rounded up to the closest 4000mm multiple, e.g., if “3000” is given then “4000” will be used)

zone_list_details: (required if “od_input_type==ZONE_LIST”)
- ***origin_zones:*** Array of
  - ***zone_id***: STRING (required only if zone_name is empty),
  - ***zone_name***: (required only if zone_id is empty),
  - ***country***: (optional, overrides “default_country” only for this element) CH (Switzerland), SE (Sweden), NL (Netherlands),
  - ***pairs_to***: (optional)
    LIST_OF_ZONES (Default, pair this origin zone to each of the zones in the “destination_zones” list);
    ALL_POSSIBLE_CONNECTIONS (pair this origin zone to all the zones –as destinations– found in the same station);

- ***destination_zones:*** Array of
  - ***zone_id***: STRING (required only if zone_name is empty),
  - ***zone_name***: (required only if zone_id is empty),
  - ***country***: (optional, overrides “default_country” only for this element) CH (Switzerland), SE (Sweden), NL (Netherlands),
  - ***pairs_to***: (optional)
    LIST_OF_ZONES (Default, pair this destination zone to each of the zones in the “origin_zones” list);
    ALL_POSSIBLE_CONNECTIONS (pair this destination zone to all the zones –as origins– found in the same station);

group_by_time (optional): STRING
ALL (default), SECONDS, MINUTES, HOURS, DAYS, WEEKS, MONTHS, QUARTERS,

group_by_time_seconds (optional): Integer, number of seconds to group by.

distribution_percentile_interval: Integer (required only for endpoint "…/distribution", input the percentile interval value in between 1 and 100 for the desired output computation.),

average_by (required only for endpoint ".../average"):

> TOTAL_MINUTES_OF_TIME_INTERVAL: total count of passenger walks divided by the number of minutes in between the end_timestamp and start_timestamp;
> PER_HOUR_OF_DAY, is the average number of passengers per hour of day (00:00:00 - 00:59:59, 01:00:00 - 01:59:59, …, 23:00:00 - 23:59:59);
> TOTAL_HOURS_OF_TIME_INTERVAL: total count of passenger walks divided by the number of hours in between the end_timestamp and start_timestamp;
> TOTAL_DAYS_OF_TIME_INTERVAL, total count of passenger walks divided by the number of days in between the end_timestamp and start_timestamp;

Output (expect JSON response):

request:
> ***ALL THE FIELDS IN THE REQUEST***

metadata:
> ***api_version***: v1

walks: (Array. If specified, one per "average_by" value in between the total time_interval, otherwise one for the complete time_interval)
> ***time_interval***:
> > ***start_timestamp***: Date (required),
> > ***end_timestamp***: Date (required),
>
> ***average_interval_value***: String (only endpoint /compute/passengers/intra-station/…/average, ej. "6" [saturday, if average_by==PER_DAY_OF_WEEK], "01:00:00 - 02:00:00" [if average_by==PER_HOUR_OF_DAY], or "61" [if average_by==TOTAL_DAYS_PER_TIME_INTERVAL, difference in between 01-nov to 31-dec])
>
> ***od_matrix***: (Array, only appears in "…/count" or "…/average" endpoints)
> > ***origin_zone_id***: STRING,
> > ***destination_zone_id***: STRING,
> > ***total_count***: Integer (only endpoint "…/count")

average: Double (only endpoint "…/average")

distribution_matrix (Array, only appears in "…/distribution" endpoints)
percentile: Integer (ex. "10", means 10th percentile, "20" means 20th percentile),
walking_distance: Double (in mm, appears only in "WALKING_DISTANCE" endpoint)
walking_time: Double (in s, appears only in "WALKING_TIME" endpoint)
walking_speed: Double (in mm/s, appears only in "WALKING_SPEED" endpoint)

density_matrix (Array, only appears in "…/density" endpoints)
cell_id: String,
density: Double (in average number passengers per cell during time interval)
walking_speed_average: Double (in mm/s per cell)

Example of endpoint:
/api/v1/compute/passengers/intra-station/zone-to-zone/count

EXAMPLE RESPONSE IN THE SWAGGER FILE FOR

- API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/count

Example 1 [Hub level user story 1; total count of passengers] In Lausanne Station, I want to know the total amount of passengers that walked from origin zones 11 and 23 to destination zones 66 and 88 between January 21 to January 27.

Input:
```
{
        “default_country”: “CH”,

        “default_parent_station”:
        {
                “stop_name”: “Lausanne”
        },

        "time_interval":{
                “start_timestamp”: “2017-01-21 00:00:00”,
                “end_timestamp”: “2017-01-27 23:59:59”
        },

        “od_input_type”: “ZONE_LIST”,
        “zone_list_details”:
        {
                “origin_zones”:
                [
                        {
                                “zone_id”: “11”,
                        },
                        {
                                “zone_id”: “23”,
                        }
                ],
                “destination_zones”:
                [
                        {
                                “zone_id”: “66”,
                        },
                        {
                                “zone_id”: “88”,
                        }
                ]
        }
}
```

Output:

```
{
        “metadata”:
        {
                “api_version”: “v1”
        },
        “request”:
        {
                “default_country”: “CH”,

                “default_parent_station”:
                {
                        “stop_name”: “Lausanne”
                },

                "time_interval":{
                        “start_timestamp”: “2017-01-21 00:00:00”,
                        “end_timestamp”: “2017-01-27 23:59:59”
                },

                “od_input_type”: “ZONE_LIST”,

                “zone_list_details”:
```

```
                {
                        “origin_zones”:
                        [
                                {
                                        “zone_id”: “11”,
                                },
                                {
                                        “zone_id”: “23”,
                                }
                        ],
                        “destination_zones”:
                        [
                                {
                                        “zone_id”: “66”,
                                },
                                {
                                        “zone_id”: “88”,
                                }
                        ]
                }
        },


        “walks”:
        [
                {
                        "time_interval": {
                                “start_timestamp”: “2017-01-21 00:00:00”,
                                “end_timestamp”: “2017-01-27 23:59:59”
                        },
                        “od_matrix”:
                        [
```

origin_zone_id	destination_zone_id	total_count
11	66	1236
11	88	3112
23	66	1334
23	88	66311

```
                        ]
                }
        ]
}
```

Example of endpoint:
/api/v1/compute/passengers/intra-station/zone-to-zone/average

EXAMPLE RESPONSE IN THE SWAGGER FILE FOR

- API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/average

Example 2 [Hub level user story 1; average count of passengers] In Lausanne Station, I want to know the daily average amount of passengers that walked from origin zones 11 and 23 to destination zones 66 and 88 between January 21 to January 27.

Input:

```
{
        “default_country”: “CH”,

        “default_parent_station”:
        {
                “stop_name”: “Lausanne”
        },

        "time_interval":{
                “start_timestamp”: “2017-01-21 00:00:00”,
                “end_timestamp”: “2017-01-27 23:59:59”
        },

        “od_input_type”: “ZONE_LIST”,

        “zone_list_details”:
        {
                “origin_zones”:
                [
                        {
                                “zone_id”: “11”,
                        },
                        {
                                “zone_id”: “23”,
                        }
                ],
                “destination_zones”:
                [
                        {
                                “zone_id”: “66”,
                        },
                        {
                                “zone_id”: “88”,
                        }
                ]
        },
        “average_by”: “TOTAL_DAYS_OF_TIME_INTERVAL”
}
```

Output:

```
{
        “metadata”:
        {
                “api_version”: “v1”
        },
         “request”:
        {
                “default_country”: “CH”,

                “default_parent_station”:
                {
                        “stop_name”: “Lausanne”
                },

                "time_interval":{
                        “start_timestamp”: “2017-01-21 00:00:00”,
                        “end_timestamp”: “2017-01-27 23:59:59”
                },

                “od_input_type”: “ZONE_LIST”,

                “zone_list_details”:
                {
                        “origin_zones”:
                        [
                                {
                                        “zone_id”: “11”,
                                },
                                {
                                        “zone_id”: “23”,
                                }
                        ],
                        “destination_zones”:
                        [
                                {
                                        “zone_id”: “66”,
                                },
                                {
                                        “zone_id”: “88”,
                                }
                        ]
                },
                “average_by”: “TOTAL_DAYS_OF_TIME_INTERVAL”
        },

        “walks”:
        [
                {
                        "time_interval":{
                                “start_timestamp”: “2017-01-21 00:00:00”,
                                “end_timestamp”: “2017-01-27 23:59:59”
                        },
                        “od_matrix”:
                        [
```

origin_zone_id	destination_zone_id	average (total_count / 7)
11	66	176.57
11	88	444.57
23	66	190.57
23	88	9473

```
                        ]
                }
        ]
}
```

Example of endpoint:
/api/v1/compute/passengers/intra-station/zone-to-zone/walking-time/distribution

EXAMPLE RESPONSE IN THE SWAGGER FILE FOR

- API endpoint:
  /api/v1/compute/passengers/intra-station/zone-to-zone/walking-time/distribution
- API endpoint:
  /api/v1/compute/passengers/intra-station/zone-to-zone/walking-distance/distribution
  (Note: instead of “walking_time” of the following example, the response matrix will output “walking_distance”.)
- API endpoint:
  /api/v1/compute/passengers/intra-station/zone-to-zone/walking-speed/distribution
  (Note: instead of “walking_time” of the following example, the response matrix will output “walking_speed”.)

Example 3 [Hub level user story 2; distribution of walking time] In Lausanne Station, I want to know the walking time distribution of passengers that walked from origin zones 11 and 23 to destination zones 66 and 88 between January 21 to January 27.

Input:

```
{
        “default_country”: “CH”,

        “default_parent_station”:
        {
                “stop_name”: “Lausanne”
        },

        "time_interval": {
                “start_timestamp”: “2017-01-21 00:00:00”,
                “end_timestamp”: “2017-01-27 23:59:59”
        },

        “od_input_type”: “ZONE_LIST”,

        “zone_list_details”:
        {
                “origin_zones”:
                [
                        {
                                “zone_id”: “11”,
                        },
                        {
                                “zone_id”: “23”,
                        }
                ],
                “destination_zones”:
                [
                        {
                                “zone_id”: “66”,
                        },
                        {
                                “zone_id”: “88”,
                        }
                ]
        },

        “distribution_precentage_interval”: 25

}
```

Output:

```
{
        “metadata”:
        {
```

```
		“api_version”: “v1”
	},
	 “request”:
	{
		“default_country”: “CH”,

		“default_parent_station”:
		{
			“stop_name”: “Lausanne”
		},

		"time_interval": {
			“start_timestamp”: “2017-01-21 00:00:00”,
			“end_timestamp”: “2017-01-27 23:59:59”
		},

		“od_input_type”: “ZONE_LIST”,

		“zone_list_details”:
		{
			“origin_zones”:
			[
				{
					“zone_id”: “11”,
				},
				{
					“zone_id”: “23”,
				}
			],
			“destination_zones”:
			[
				{
					“zone_id”: “66”,
				},
				{
					“zone_id”: “88”,
				}
			]
		},

		“distribution_precentage_interval”: 25

	},

	“walks”:
	[
		{
			"time_interval":{
				“start_timestamp”: “2017-01-21 00:00:00”,
				“end_timestamp”: “2017-01-27 23:59:59”
```

```
                        },
                        “distribution_matrix”:
                        [
```

percentile	walking_time
0	210
25	245
50	248
75	332
100	425

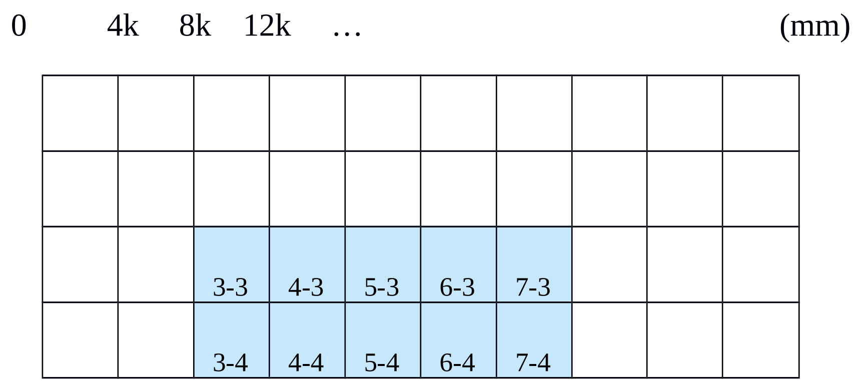
```
                        ]
                }
        ]
}
```

Example of endpoint:
/api/v1/compute/passengers/intra-station/zone-to-zone/density

EXAMPLE RESPONSE IN THE SWAGGER FILE FOR

- API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/density
- API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/walking-speed/average/density *(Note: instead of “passenger_density” of the following example, the response matrix will output “walking_speed_average”.)*

Example 4 [Hub level user story 5; grid; passenger density] In Lausanne station, I want to know the density (= number of passengers) in each grid cell in each time interval of 30 seconds; on January 21 between 08:00:00 and 08:09:59. The grid cells are 4000mm x 4000mm sqares.



		3-5	4-5	5-5	6-5	7-5			

Input:

```
{
        “default_country”: “CH”,

        “default_parent_station”:
        {
                “stop_name”: “Lausanne”
        },

        "time_interval": {
                “start_timestamp”: “2017-01-21 08:00:00”,
                “end_timestamp”: “2017-01-21 08:10:00”
        },

        “od_input_type”: “GRID”,

        “grid_details”:
        {
                “grid_boundaries”:
                {
                        “corner”:
                        {
                                “x”: 0,
                                “y”: 0
                        },
                        “length”: 4000,
                        “width”: 4000
                }
        },
        “group_by_time_seconds”: 30
}
```

Output:

```
{
	“metadata”:
	{
		“api_version”: “v1”
	},
	 “request”:
	{
		“default_country”: “CH”,

		“default_parent_station”:
		{
			“stop_name”: “Lausanne”
		},

		"time_interval": {
			“start_timestamp”: “2017-01-21 08:00:00”,
			“end_timestamp”: “2017-01-21 08:10:00”
		},

		“od_input_type”: “GRID”,

		“grid_details”:
		{
			“grid_boundaries”:
			{
				“corner”:
				{
					“x”: 0,
					“y”: 0
				},
				“length”: 4000,
				“width”: 4000
			}
		},

		“group_by_time_seconds”: 30

	},

	“walks”:
	[
		{
			"time_interval":{
				“start_timestamp”: “2017-01-21 08:00:00”,
				“end_timestamp”: “2017-01-21 08:00:30”
			},
```

“density _matrix”:
[

cell_id	density
3-3	1.23
4-3	1.47
…	…
7-5	0.25

]
},
{
"time_interval":{
“start_timestamp”: “2017-01-21 08:00:30”,
“end_timestamp”: “2017-01-21 08:01:00”
},

“density_matrix”:
[

cell_id	density
3-3	1.11
4-3	1.15
…	…
7-5	0.77

]
},

{
… time intervals in between 08:01:00 and 08:09:30
},

{
"time_interval": {
“start_timestamp”: “2017-01-21 08:09:30”,
“end_timestamp”: “2017-01-21 08:10:00”
},

“density_matrix”:
[

cell_id	passenger_density
3-3	1.00
4-3	1.11
…	…
7-5	0.33

]
}

```
        ]
}
```

4.2.9. API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/average

Check API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/count

Example 2

4.2.10. API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/walking-distance/distribution

Check API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/count

Example 3 (replace the output “walking_time” by “walking_distance”)

4.2.11. API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/walking-time/distribution

Check API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/count

Example 3

4.2.12. API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/walking-speed/distribution

Check API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/count

Example 3 (replace the output “walking_time” to “walking_speed”)

4.2.13. API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/density

Check API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/count

Example 4

4.2.14. API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/walking-speed/average/density

Check API endpoint: /api/v1/compute/passengers/intra-station/zone-to-zone/count

Example 4 (replace the output “passenger_density” to “walking_speed_average_density”)

4.3. Potential future (computation) API extensions

4.3.1. API endpoint: /api/v1/compute/passengers/transfer/count (extension 1)

To make the API more "user-friendly-ish", the input request JSON would need to specify the "trip_id" (something like: ch_trip_544523, where let say "544523" is the unique identifier inside Switzerland for that trip in the trips.txt), because we can't identify an specific trip node with only the route_short_name (ex. "IC70112") as an input. The "IC70112" label represents all the trips running daily on the same route/path or the rounte_short_name.

- So one case is that the person who is going to use this specific api endpoint already knows, somehow, the ***time-specific*** trip_id (ch_trip_544523), and the use this as input/output (Note: this is the original version case).
- But a second case (the maybe "more user-friendly"), is that the person only knows the route_short_name (IC70112), and the "scheduled_arrival time" (08:00:00) of the "transfer_stop" (which is required as an input too), and with these 2 input parazone_alias we can deduce the time-specific "trip_id" we need (ch_trip_544523) and the possible connecting outgoing trips.

<u>Input:</u>

```
{
        “start_timestamp”: “2017-01-26 00:00:00”,
        “end_timestamp”: “2017-01-29 23:59:59”,
         “incoming_journey_legs”: [
                {
                        “route_short_name”: “IC70112”,
                        “scheduled_arrival time”: “08:00:00”
                        //IMPLICIT// “destination_stop_name”: SEE “transfer_stops”
                },
        ],
        “transfer_stops”:
        [
                {
                        “transfer_stop_id”: “ ” OR “transfer_stop_name”: “Zürich HB”;
                }
        ]
        “outgoing_journey_legs”: [
                {
                        //IMPLICIT// “origin_stop_name”: SEE “transfer_stops”
                        “route_short_name”: “IC70155”,
                        “scheduled_departurel time”: “08:15:00”
                }
        ],
}
```

4.3.2. API endpoint: /api/v1/compute/passengers/transfer/count (extension 2)

Get the number of passengers who went from a list of specific incoming journey_legs (asking origin_stop_name OR origin_stop_id), then stopped and transferred in a specific stop (transfer_stop_id), to then continue their journeys to a list of outgoing journey_legs (asking destination_stop_name OR destination_stop_id)

Example: Count of passenger who went from stops {“Zürich Altstetten” (=stopID1) or “Rüschlikon” (=stopIDID3), in direction to {“Bern HB” (=stopID4), “Basel HB” (=stopID5), or “Genève” (=stopID6)}, in between Jan 26 to Jan 29.

Input:

```
{
        “start_timestamp”: “2017-01-26 00:00:00”,
        “end_timestamp”: “2017-01-29 23:59:59”,
         “incoming_journey_legs”: [
                {
                        “origin_stop_id”: “Zürich Altstetten”,
                        //IMPLICIT// “destination_stop_name”: SEE “transfer_stops”
                },
                {
                        “origin_stop_name”: “Rüschlikon”,
                        //IMPLICIT// “destination_stop_name”: SEE “transfer_stops”
                }
        ],
        “transfer_stops”:
        [
                {
                        “transfer_stop_id”: “ ” OR “transfer_stop_name”: “Zürich HB”;
                }
        ]
        “outgoing_journey_legs”: [
                {
                        //IMPLICIT// “origin_stop_name”: SEE “transfer_stops”
                        “destination_stop_name”: “Bern HB”
                },
                {
                        //IMPLICIT// “origin_stop_name”: SEE “transfer_stops”
                        “destination_stop_name”: “Basel HB”
                },
                {
                        //IMPLICIT// “origin_stop_name”: SEE “transfer_stops”
                        “destination_stop_name”: “Genève”
```

```
		}
	],
}
```

Output:

```
{

	“request”: {
		SAME AS ORIGINAL REQUEST
	},
	“journeys”: [
		{
			“start_timestamp”: “2017-01-26 00:00:00”,
			“end_timestamp”: “2017-01-29 23:59:59”,
			“transfer_stop_name”: “Zürich HB”,
			“od_matrix”: [
```

incoming_journey_leg origin_stop_name	**outgoing _journey_leg destination_stop_name**	**total_count**	**Average (total_count / time interval)**
Zürich Altstetten	Bern HB	523	523 / 4 (per day)
Zürich Altstetten	Basel HB	661	..
Zürich Altstetten	Genève	345	
Rüschlikon	Bern HB	122	
Rüschlikon	Basel HB	10234	
Rüschlikon	Genève	33124	

```
			]
		}

	]
}
```

4.3.3. API endpoint: /api/v1/compute/routes/count

Description: get an OD-Matrix with the total number of passengers, within the time frame specified in the request, from all the stop-to-stop sequences for every ROUTE specified. This API endpoint doesn’t consider passenger transfers or connections between routes (i.e. there are no two connected journey_legs in passengers journeys, but only one journey_leg for every journey a passenger travels). This endpoint can be used to calculate the passengers count of single routes. The count value is the aggregation (sum) of every trip taken place during the time frame in a specific route.

HTTP Verb: POST

Input (send JSON request):

```
    start_timestamp: Date (required),
    end_timestamp: Date (required),
    routes: (Array, if empty use default = ALL)
        route_id: STRING (optional, if empty use default=transverse all the stops path
        from the origin_stop_id (if provided) to all its possible destination, or from all
        the possible origin stops to the destination_stop_id (if provided), or if both are
        provided then compute only from origin_stop_id to destination_stop_id in this
        route),
        origin_stop_id: STRING (optional, if empty use default=first stop in trip
        sequence),
        destination_stop_id: STRING (optional, if empty use default=last stop in trip
        sequence),
        show_intermediate_stops: BOOLEAN (optional, TRUE (default), get all
        intermediate stops in between origin_stop_id and destination_stop_id; FALSE,
        aggregate all the stops from origin_stop_id and destination_stop_id to a single
        count)
    group_by_time (optional): STRING - ALL (default), SECONDS, MINUTES,
    HOURS, DAYS, WEEKS, MONTHS, QUARTERS
```

Output (expect JSON response):

```
    request:
        ALL THE FIELDS IN THE REQUEST
    od_matrix: (Array)
        start_timestamp: Date,
        end_timestamp: Date,
        routes: (Array)
            route_id: STRING,
            origin_stop_id: STRING,
            destination_stop_id: STRING,
            count: Integer
```

Example 1. I want to know the passenger count for every day in between 2017-01-21 and 2017-01-27, on the routes "Line 1" (starting on the stop "S3") until the last stop, and route "Line 3" for the whole trip (first to last stop).

Input:

```
{
        "start_timestamp": "2017-01-21 00:00:00",
        "end_timestamp": "2017-01-27 23:59:59",
        "routes": [
                {
                        "route_id": "Line 1",
                        "origin_stop_id": "S3"
                },
                {
                        "route_id": "Line 3"
                },
        ],
        "group_by_time": "DAYS"
}
```

Output:

```
{

        "request": {
                "start_timestamp": "2017-01-21 00:00:00",
                "end_timestamp": "2017-01-27 23:59:59",
                "routes": [
                        {
                                "route_id": "Line 1",
                                "origin_stop_id": "S3"
                        },
                        {
                                "route_id": "Line 3"
                        },
                ],
                "group_by_time": "DAYS"
        },
        "od_matrix": [
                {
                        "start_timestamp": "2017-01-21 00:00:00",
                        "end_timestamp": "2017-01-21 23:59:59",
                        "routes": [
```

route_id	**origin_stop_id**	**destination_stop_id**	**count**
Line 1	S3	S4	3
Line 1	S4	S5	4

Line 1	…intermediate stops of Line 1	...	…
Line 1	S8	S9 (last stop L1)	5
Line 3	S17 (first stop L3)	S18	22
Line 3	S18	S19	12
Line 3	S19	S20	17
Line 3	S20	S21 (last stop L3)	13

]
},
{
“start_timestamp”: “2017-01-22 00:00:00”,
“end_timestamp”: “2017-01-22 23:59:59”,
“routes”: [

route_id	**origin_stop_id**	**destination_stop_id**	**count**
Line 1	S3	S4	2
Line 1	S4	S5	5
Line 1	…intermediate stops of Line 1	...	…
Line 1	S8	S9 (last stop L1)	2
Line 3	S17 (first stop L3)	S18	12
Line 3	S18	S19	32
Line 3	S19	S20	27
Line 3	S20	S21 (last stop L3)	13

]
},
{
… // OD-Matrixes from 2017-01-23 to 2017-01-26, one per day // …
},
{
“start_timestamp”: “2017-01-22 00:00:00”,
“end_timestamp”: “2017-01-22 23:59:59”,
“routes”: [

route_id	**origin_stop_id**	**destination_stop_id**	**count**
Line 1	S3	S4	2
Line 1	S4	S5	8
Line 1	…intermediate stops of Line 1	...	…
Line 1	S8	S9 (last stop L1)	3
Line 3	S17 (first stop L3)	S18	12
Line 3	S18	S19	2
Line 3	S19	S20	7
Line 3	S20	S21 (last stop L3)	19

]

}
]
}

Example 2. I want to know the total passenger count in between 2017-02-01 and 2017-02-05, on the routes "Line 1", "Line 3", and "Line 5".

Input:

```
{
        "start_timestamp": "2017-02-01 00:00:00",
        "end_timestamp": "2017-02-05 23:59:59",
        "routes": [
                {
                        "route_id": "Line 1"
                },
                {
                        "route_id": "Line 3"
                },
                {
                        "route_id": "Line 5"
                }

        ],
        "group_by_time": "ALL"
}
```

Output:

```
{

        "request": {
                "start_timestamp": "2017-02-01 00:00:00",
                "end_timestamp": "2017-02-05 23:59:59",
                "routes": [
                        {
                                "route_id": "Line 1"
                        },
                        {
                                "route_id": "Line 3"
                        },
                        {
                                "route_id": "Line 5"
                        }

        ],
        "group_by_time": "ALL"

        },
        “od_matrix”:
        [
                {
                        “start_time”: “2017-02-01 00:00:00”,
                        “end_time”: “2017-02-05 23:59:59”,
                        “routes”:
                        [
```

route_id	origin_stop_id	destination_stop_id	count
Line 1	S1 (first stop L1)	S2	32342
Line 1	S2	S3	41233
Line 1	… intermediate stops of Line 1	…	…
Line 1	S8	S9 (last stop L1)	5338
Line 3	S17 (first stop L3)	S18	22127
Line 3	S18	S19	12215
Line 3	S19	S20	1721
Line 3	S20	S21 (last stop L3)	13443
Line 5	S65 (first stop L5)	S68	2211
Line 5	S68	S76	32379
Line 5	S76	S99	1421
Line 5	S99	S101	1212
Line 5	…intermediate stops of Line 5	…	
Line 5	S112	S114 (last stop L5)	124

```
                        ]
                }
        ]
}
```

Example 3. I want to know the total passenger count, aggregated from the first stop to the last stop, in between 2017-02-01 and 2017-02-05, on the routes “Line 1”, “Line 3”, and “Line 5”.

<u>*Input:*</u>

```
{
        “start_timestamp”: “2017-02-01 00:00:00”,
        “end_timestamp”: “2017-02-05 23:59:59”,
        “routes”: [
                {
                        “route_id”: “Line 1”,
                        “show_intermediate_stops”: “FALSE”
                },
                {
                        “route_id”: “Line 3”,
				“show_intermediate_stops”: “FALSE”

			},
			{
				“route_id”: “Line 5”,
				“show_intermediate_stops”: “FALSE”
			}

		],
		“group_by_time”: “ALL”,

}
```

Output:

```
{

		“request”: {
			“start_timestamp”: “2017-02-01 00:00:00”,
			“end_timestamp”: “2017-02-05 23:59:59”,
			“routes”: [
				{
					“route_id”: “Line 1”,
					“show_intermediate_stops”: “FALSE”
				},
				{
					“route_id”: “Line 3”,
					“show_intermediate_stops”: “FALSE”

				},
				{
					“route_id”: “Line 5”,
					“show_intermediate_stops”: “FALSE”
				}

		],
		“group_by_time”: “ALL”

		},
		“od_matrix”:
		[
			{
				“start_time”: “2017-02-01 00:00:00”,
				“end_time”: “2017-02-05 23:59:59”,
				“routes”:
				[
```

route_id	**origin_stop_id**	**destination_stop_id**	**count**
Line 1	S1 (first stop L1)	S9 (last stop L1)	733242

Line 3	S17 (first stop L3)	S21 (last stop L3)	392127
Line 5	S65 (first stop L5)	S114 (last stop L5)	842211

```
                        ]
                ]
        ]
}
```

4.3.4. API endpoint: /api/v1/compute/routes/average

Description: get an OD-Matrix with the average number of passengers, within the time frame specified in the request, from all the stop-to-stop sequences for every ROUTE specified. This API endpoint doesn't consider transfers or connections between routes (i.e. there are no two connected journey_legs in passengers journeys, but only one journey_leg per journey), only single routes passengers count).

The average value is calculated by the "average_by" property:

- TRIPS, is the average number of passenger during the time frame within a route from stop-to-stop (= total number of passengers from-stop-to-stop divided by number of trips);
- HOURS_PER_DAY, is the average number of passengers per hour of day (00:00:00 - 00:59:59, 01:00:00 - 01:59:59, …, 23:00:00 - 23:59:59);
- DAYS_PER_WEEK, average per day of week (MONDAYS, TUESDAYS, …, SUNDAY);
- DAYS_PER_MONTH, average per day of week (1st, 2nd, …, 31st);
- WEEKS_PER_MONTH, average per week in month (week 1, week 2, week 3, week 4);
- WEEKS_PER_YEAR, average per week in a year (week 1, week 2, …, week 52);
- MONTHS_PER_YEAR, average per month in a year (January, February, …, December);
- QUARTERS_PER YEAR, average per quarter in a year (Q1, Q2, Q3, Q4);

HTTP Verb: POST

Input (send JSON request):

start_timestamp: Date (required),
end_timestamp: Date (required),
routes: (Array, if empty use default = ALL)
    route_id: STRING (optional, if empty use default=transverse all the stops path from the origin_stop_id (if provided) to all its possible destination, or from all the possible origin stops to the destination_stop_id (if provided), or if both are provided then compute only from origin_stop_id to destination_stop_id in this route),
    origin_stop_id: STRING (optional, if empty use default=first stop in trip sequence),
    destination_stop_id: STRING (optional, if empty use default=last stop in trip sequence),
    show_intermediate_stops: BOOLEAN (optional, TRUE (default), get all intermediate stops in between origin_stop_id and destination_stop_id; FALSE, aggregate all the stops from origin_stop_id and destination_stop_id to a single count)
average_by: TRIPS, HOURS_PER_DAY, DAYS_PER_WEEK, DAYS_PER_MONTH, WEEKS_PER_MONTH, WEEKS_PER_YEAR, MONTHS_PER_YEAR, QUARTERS_PER YEAR

Output (expect JSON response):

```
        request:
                ALL THE FIELDS IN THE REQUEST
        od_matrix: (Array)
                start_timestamp: Date,
                end_timestamp: Date,
                routes: (Array)
                        route_id: STRING,
                        origin_stop_id: STRING,
                        destination_stop_id: STRING,
                        average: Double
```

Example 4. I want to know the average passenger count per hour of the day in between 2017-01-21 and 2017-01-27, on the routes "Line 1" (starting on the stop "S3") until the last stop, and route "Line 3" aggregated from the first to last stop.

Input:

```
{
        "start_timestamp": "2017-01-21 00:00:00",
        "end_timestamp": "2017-01-27 23:59:59",
        "routes": [
                {
                        "route_id": "Line 1",
                        "origin_stop_id": "S3"
                },
                {
                        "route_id": "Line 3",
                        "show_intermediate_stops": "FALSE"
                },
        ],
        "average_by": "HOURS_PER_DAY"
}
```

Output:

```
{

        "request": {
                "start_timestamp": "2017-01-21 00:00:00",
                "end_timestamp": "2017-01-27 23:59:59",
                "routes": [
                        {
                                "route_id": "Line 1",
                                "origin_stop_id": "S3"
                        },
            {
                “route_id”: “Line 3”,
                “show_intermediate_stops”: “FALSE”
            },
        ],
        “average_by”: “HOURS_PER_DAY”
    },
    “od_matrix”: [
        {
            “start_timestamp”: “00:00:00”,
            “end_timestamp”: “00:59:59”,
            “routes”: [
```

route_id	origin_stop_id	destination_stop_id	average
Line 1	S3	S4	1
Line 1	S4	S5	2
Line 1	… intermediate stops of Line 1	...	…
Line 1	S8	S9 (last stop L1)	2
Line 3	S17 (first stop L3)	S21 (last stop L3)	324

```
            ]
        },
        {
            “start_timestamp”: “01:00:00”,
            “end_timestamp”: “01:59:59”,
            “routes”: [
```

route_id	origin_stop_id	destination_stop_id	average
Line 1	S3	S4	4
Line 1	S4	S5	3
Line 1	…intermediate stops of Line 1	...	…
Line 1	S8	S9 (last stop L1)	1
Line 3	S17 (first stop L3)	S21 (last stop L3)	234

```
            ]
        },
        {
            … // OD-Matrixes from 02:00:00 to 22:59:59, one per hour of day// …
        },
        {
            “start_timestamp”: “23:00:00”,
            “end_timestamp”: “23:59:59”,
            “routes”: [
```

route_id	origin_stop_id	destination_stop_id	average
Line 1	S3	S4	2
Line 1	S4	S5	8
Line 1	…intermediate stops of Line 1	...	…
Line 1	S8	S9 (last stop L1)	3

Line 3	S17 (first stop L3)	S21 (last stop L3)	341

```
                              ]

                    }
          ]
}
```

Example 5. I want to know the average trip passenger count in between 2017-02-01 and 2017-02-05, on the routes "Line 1", "Line 3", and "Line 5".

Input:

```
{
          "start_timestamp": "2017-02-01 00:00:00",
          "end_timestamp": "2017-02-05 23:59:59",
          "routes": [
                    {
                              "route_id": "Line 1",
                    },
                    {
                              "route_id": "Line 3",
                    },
                    {
                              "route_id": "Line 5",
                    }

          ],
          "average_by": "TRIPS"
}
```

Output:

```
{

          "request": {
                    "start_timestamp": "2017-02-01 00:00:00",
                    "end_timestamp": "2017-02-05 23:59:59",
                    "routes": [
                              {
                                        "route_id": "Line 1"
                              },
                              {
                                        "route_id": "Line 3"
                              },
                              {
```

```
					“route_id”: “Line 5”
				}

		],
		“average_by”: “TRIPS”

		},
		“od_matrix”:
		[
			{
				“start_time”: “2017-02-01 00:00:00”,
				“end_time”: “2017-02-05 23:59:59”,
				“routes”:
				[
```

trip_id	**origin_stop_id**	**destination_stop_id**	**average**
Line 1	S1 (first stop L1)	S2	12
Line 1	S2	S3	32
Line 1	S…	S...	…
Line 1	S8	S9 (last stop L1)	15
Line 3	S17 (first stop L3)	S18	32
Line 3	S18	S19	33
Line 3	S19	S20	34
Line 3	S20	S21 (last stop L3)	12
Line 5	S65 (first stop L5)	S68	23
Line 5	S68	S76	11
Line 5	S76	S99	22
Line 5	S99	S101	23
Line 5	S…	S…	
Line 5	S112	S114 (last stop L5)	23

```
				]
			}
		]
}
```

5. Conclusions and Prospects

This document has described the design and implementation of the Big Travel Demand Analytics Support Tool. The tool has been conceived as a web application running on the IBM Bluemix cloud platform-as-a-service.

The data storage core of the application is a graph database. The graph database implements a novel data model for the combination of transit and passenger data, which has been developed especially for this project in order to enable an analysis of complex travel demand data. The main innovation of this data model is to combine transit data, for which a de-facto standard already existed (GTFS) albeit only in relational form, with passenger data, for which no standard was available so far, and to combine these two layers into a combined transit-and-passenger data model. Furthermore, the data model was extended to include hub-level passenger data to capture the walking behavior inside transit hubs.

The Big Travel Demand Analytics Support Tool provides its analysis functionality via a set of well-defined APIs. These APIs can be called from any consuming program in order to perform further processing. One main use of the APIs is the visualization of travel demand data, thus the tool will feed into the Visualization Tool developed as D1.3.

The use of the APIs, either directly or via the Visualization Tool, during the case studies will serve as a test of whether the new concept of a graph database is suitable for storing and analysing travel demand data. In particular, the performance of the graph queries, implemented in Gremlin, will be tested. Given the limited experience in the field of graph databases so far, this will serve as an interesting benchmark and test case.

In order to run the tool outside the IBM Bluemix, a certain adjustments will have to be done. Since IBM Graph is only available as a service on Bluemix, another standalone graph database must be chosen and set up in order to implement the same data model. This graph database must provide Gremlin APIs. Furthermore, a webserver running the Node.js runtime environment has to be set up. The analysis APIs should only need minor configuration adjustments to connect to the new database, but other than that should be interoperable due to the use of Gremlin.

Appendix A: Modifications of API specifications

/api/v1/compute/passengers/stop-to-stop/count:
/api/v1/compute/passengers/stop-to-stop/average:

Options not implemented:

od_matrix_output: (optional) SAME_VALUE_AS_OD_LISTS (Default, returns the value(s) specified in the "origin_stops" and "destination_stops" lists), STOP_ID (return only the stop_id value in the OD-Matrix), STOP_NAME (return only the stop_name value in the OD-Matrix), STOP_ID_AND_STOP_NAME (return the stop_id and stop_name value in the OD-Matrix)

group_by_time (optional): STRING
ALL (default), SECONDS, MINUTES, HOURS, DAYS, WEEKS, MONTHS, QUARTERS,

average_by (only for endpoint /compute/passengers/stop-to-stop/average):
PER_DAY_OF_WEEK, average per day of the week (1 –monday–, 2, …, 7);
PER_DAY_OF_MONTH, average per day of the month (1, 2, 3, 4,…, 30, 31);
PER_WEEK_OF_MONTH, average per week in month (1 –week–, 2, 3, 4);
PER_WEEK_OF_YEAR, average per week in a year (1, 2, …, 53);
TOTAL_WEEKS_OF_TIME_INTERVAL: total count of passenger journeys divided by the number of weeks in between the end_timestamp and start_timestamp;
PER_MONTH_OF_YEAR, average per month in a year (1-january, …, 12-december);
TOTAL_MONTHS_OF_TIME_INTERVAL: total count of passenger journeys divided by the number of months (30 days/month) in between the end_timestamp and start_timestamp;
PER_QUARTER_OF_YEAR, average per quarter in a year (1 –Q–, 2, 3, 4);
[TOTAL_QUARTERS_OF_TIME_INTERVAL] total count of passenger journeys divided by the number of quarters (4 quarters/year) in between the end_timestamp and start_timestamp;
[PER_YEARS] average per years (2017, 2018, 2019,...);
[TOTAL_YEARS_OF_TIME_INTERVAL] total count of passenger journeys divided by the number of years (365 days/year) in between the end_timestamp and start_timestamp.

/api/v1/compute/passengers/route-to-route/count:
/api/v1/compute/passengers/route-to-route/average:

Options not implemented:

transfer_stops:
stop_name: STRING (required if “stop_id” is empty),

average_by (required only for endpoint …route-to-route/average):
PER_HOUR_OF_DAY, is the average number of passengers per hour of day (00:00:00 - 00:59:59, 01:00:00 - 01:59:59, …, 23:00:00 - 23:59:59);
PER_DAY_OF_WEEK, average per day of the week (1 –monday–, 2, …, 7);
PER_DAY_OF_MONTH, average per day of the month (1, 2, 3, 4,…, 30, 31);
PER_WEEK_OF_MONTH, average per week in month (1 –week–, 2, 3, 4);
PER_WEEK_OF_YEAR, average per week in a year (1, 2, …, 53);
TOTAL_WEEKS_OF_TIME_INTERVAL: total count of passenger journeys divided by the number of weeks in between the end_timestamp and start_timestamp;
PER_MONTH_OF_YEAR, average per month in a year (1-january, …, 12-december);
TOTAL_MONTHS_OF_TIME_INTERVAL: total count of passenger journeys divided by the number of months (30 days/month) in between the end_timestamp and start_timestamp;
PER_QUARTER_OF_YEAR, average per quarter in a year (1 –Q–, 2, 3, 4);
[TOTAL_QUARTERS_OF_TIME_INTERVAL] total count of passenger journeys divided by the number of quarters (4 quarters/year) in between the end_timestamp and start_timestamp;
[PER_YEARS] average per years (2017, 2018, 2019,...);
[TOTAL_YEARS_OF_TIME_INTERVAL] total count of passenger journeys divided by the number of years (365 days/year) in between the end_timestamp and start_timestamp.

/api/v1/compute/passengers/intra-station/zone-to-zone/count:
/api/v1/compute/passengers/intra-station/zone-to-zone/average:
/api/v1/compute/passengers/intra-station/zone-to-zone/walking-distance/distribution
/api/v1/compute/passengers/intra-station/zone-to-zone/walking-speed/distribution
/api/v1/compute/passengers/intra-station/zone-to-zone/walking-time/distribution

Options not implemented:

zone_list_details: (required if "od_input_type==ZONE_LIST")
  origin_zones: Array of
    zone_name: (required only if zone_id is empty),
    country: (optional, overrides "default_country" only for this element)
    CH (Switzerland), SE (Sweden), NL (Netherlands),
    pairs_to: (optional)
    LIST_OF_ZONES (Default, pair this origin zone to each of the zones in the "destination_zones" list);
    ALL_POSSIBLE_CONNECTIONS (pair this origin zone to all the zones –as destinations– found in the same station);

  destination_zones: Array of
    zone_name: (required only if zone_id is empty),
    country: (optional, overrides "default_country" only for this element)
    CH (Switzerland), SE (Sweden), NL (Netherlands),
    pairs_to: (optional)
    LIST_OF_ZONES (Default, pair this destination zone to each of the zones in the "origin_zones" list);
    ALL_POSSIBLE_CONNECTIONS (pair this destination zone to all the zones –as origins– found in the same station);

group_by_time (optional): STRING
  ALL (default), SECONDS, MINUTES, HOURS, DAYS, WEEKS, MONTHS, QUARTERS,

group_by_time_seconds (optional): Integer, number of seconds to group by.

average_by (required only for endpoint ".../average"):
  PER_MINUTE_OF_HOUR, is the average number of passengers per hour of day (00:00:00 - 00:00:59, 00:01:00 - 00:01:59, …, 23:59:00 - 23:59:59);
  PER_DAY_OF_WEEK, average per day of week (1 –monday–, 2, …, 7);
  PER_DAY_OF_MONTH, average per day of week (1, 2, 3, 4,…, 30, 31);
  PER_WEEK_OF_MONTH, average per week in month (1 –week–, 2, 3, 4);
  PER_WEEK_OF_YEAR, average per week in a year (1, 2, …, 53);
  TOTAL_WEEKS_OF_TIME_INTERVAL: total count of passenger walks divided by the number of weeks in between the end_timestamp and start_timestamp;
  PER_MONTH_OF_YEAR, average per month in a year (1-january, …, 12-december);

TOTAL_MONTHS_OF_TIME_INTERVAL: total count of passenger walks divided by the number of months (30 days/month) in between the end_timestamp and start_timestamp;
PER_QUARTERS_OF_YEAR, average per quarter in a year (1 –Q–, 2, 3, 4);
[TOTAL_QUARTERS_OF_TIME_INTERVAL] total count of passenger journeys divided by the number of quarters (4 quarters/year) in between the end_timestamp and start_timestamp;
[PER_YEARS] average per years (2017, 2018, 2019,...);
[TOTAL_YEARS_OF_TIME_INTERVAL] total count of passenger journeys divided by the number of years (365 days/year) in between the end_timestamp and start_timestamp.

/api/v1/compute/passengers/intra-station/zone-to-zone/density:
/api/v1/compute/passengers/intra-station/zone-to-zone/walking-speed/average/density:

Options not implemented:

grid_details: (required only if “od_input_type==GRID”)
all_cells_in_station: BOOLEAN (optional)
False (Default, if false specify the “grid_boundaries” parameter),
True (if true, all the grid cells inside the station is used to make the computation).