# OpenTrainCommunity Documentation

***Release 1.0.0***

**Maaike**

**Dec 24, 2018**

# Overview

# CHAPTER 1

## Overview of the OpenTrain project

tl; dr - We apply technology to enhance democracy and to empower Israeli citizens. This is our zionism.

This project aims to make Israel Railways punctuality data accessible, simple and understandable for everyone. Israel Railways is a state-owned company, and we believe that as citizens, it is our right to be able to understand how well this service is functioning.

Unfortunately, not much data is put forth by the Israel Railways company as publicly accessible information. We have received punctuality data from the Israel Railways company directly and are in the process of transforming, analysing and displaying that data at http://otrain.org, in a simple and understandable way.

We are a group of volunteers - programmers, designers, data analysts and content writers – that get together every week to achieve that goal.

CHAPTER 2

# General Info

**Website**: http://otrain.org

**Website code**: https://github.com/hasadna/OpenTrainCommunity

**Hosting is at DigitalOcean. The user is**: hasadna.opentrain@gmail.com

**Server API (client is Angular)**: http://otrain.org/api/docs/

CHAPTER 3

# Opening the django shell on the server

Do the following:

1. ssh to the server (ask us for instructions).
2. cd /home/opentrain/work/OpenTrainCommunity/simple/train
3. opentrain@otdata: ~/work/OpenTrainCommunity/simple/train$ python manage.py shell_plus

CHAPTER 4

# Installing the website

Follow the instructione here:

https://github.com/hasadna/OpenTrainCommunity

# CHAPTER 5

# Our data

We take the raw Israel Railways data we receive and do minimal processing - mainly structuring it and marking invalid data as such. Each data point in our database has a pointer to a specific line in one of the raw files we've received from Israel Railways. That way, if and when some question arises about the data, we can always pinpoint the exact source of that data.

- Raw Excel data received from Israel Railways:
- http://otrain.org/files/xl-feb-2016/times/
- http://otrain.org/files/xl-april-2016/
- Dump of our database: http://otrain.org/files/dumps
- Our database in csv format: http://otrain.org/files/dumps-csv

## 5.1 Adding new data

- Put the data in ~/public_html/files/ on the server in an appropriate folder (follow the standard there).
- While under 'workon train2' virtualenv and in the ~/work/OpenTrainCommunity/train2 folder, run:

```
python manage.py parsexl /home/opentrain/public_html/files/xl-2016-nov/xl-2016-nov.
↪xlsx
```

Make sure to change to your excel file. You should get something similar to:

```
[28/11/2016 17:30:20] INFO [utils_2015:137] Creating routes
[28/11/2016 17:31:14] INFO [utils_2015:141] # of valid trips = 9592
[28/11/2016 17:31:14] INFO [utils_2015:142] # of invalid trips = 136
[28/11/2016 17:31:14] INFO [utils_2015:146] Reason: sample has different planned and␣
↪stopped count = 29
[28/11/2016 17:31:14] INFO [utils_2015:146] Reason: missing actual_arrival count = 27
[28/11/2016 17:31:14] INFO [utils_2015:146] Reason: missing actual_departure count =␣
↪57
```

(continues on next page)

(continued from previous page)

```
[28/11/2016 17:31:14] INFO [utils_2015:146] Reason: first stop is not is_source count␣
↪= 21
[28/11/2016 17:31:14] INFO [utils_2015:146] Reason: last stop is not is_dest count = 2
```

# CHAPTER 6

## Database tables schema

| data_sample | represents arrival, departure at a station, time as part of a trip |
|---|---|
| id | automatic ID by DB |
| index | the index of the stop in the trip[1] |
| gtfs_stop_id | the station GTFS id |
| stop_id | the station id, a foreign key to the data_stops table |
| valid | data sanity check |
| invalid_reason | description of invalid reason, if invalid |
| is_source | whether it the first passengers stop (there may be non-passenger stops before) |
| is_dest | whether it the last passengers stop |
| actual_arrival | time of arrival |
| actual_arrival_fixed | is field actual_arrival missing, if so - used exp_arrival |
| exp_arrival | the planned time |
| delay_arrival | the delta of actual_arrival – exp_arrival |
| actual_departure | time of departure |
| actual_departure_fixed | is field actual_departure missing, if so - used exp_departure |
| exp_departure | planned departure |
| delay_departure | the delta of actual_departure – exp_departure |
| filename | source of data (for debugging purposes) |
| line_number | the line in that file (for debugging purposes) |
| sheet_idx | the sheet in that file (for debugging purposes) |
| trip_id | the id of the trip <train, date> (train = route id) |

[1] Note that there are gaps in the indexes since the original indexing includes operational stops.

| data_trip | collection of samples representing a unique trip from source to destination |
|---|---|
| id | the trip id, a non generated primary key (timestamp & train nr) |
| train_num | train num as given by the train |
| date | date of trip |
| valid | data sanity check |
| invalid_reason | description of invalid reason, if invalid |
| x_week_day_local | day of week (0 to 6) (first sample in the trip) |
| x_hour_local | expected hour of departure (first sample in the trip) |
| route_id | foreign key to the route table |
| x_avg_delay_arrival | average delay over all samples in the trip |
| x_cache_version | cache version being used for this table, used for table updates |
| x_max2_delay_arrival | second largest delay among the route's samples |
| x_max_delay_arrival | largest delay among the route's samples |
| x_before_last_delay_arrival | delay at route's second to last sample |
| x_last_delay_arrival | delay at route's last sample |

| data_route | list of stops for a repeating route |
|---|---|
| id | automatic ID by DB |
| stop_ids | json list of stop ids |

| data_stop | static info about stops in the rail network |
|---|---|
| id | automatic ID by DB |
| gtfs_stop_id | gtfs (General Transit Feed Specification) station id |
| english | english name |
| hebrews | hebrew names (json list) |
| lat | latitude |
| lon | longitude |

CHAPTER 7

# Connecting to the db directly

This requires installing postgres and using psql, a postgres sql client.:

```
psql -h 104.131.88.144 --user guest --dbname train2
```

The password is guest

CHAPTER 8

# Accessing the data using redash

You can use the redash website to run SQLs and preview the result.

You can share the results with others.

Pretty cool tool! Check it out:

http://hasadna.redashapp.com

Note that it requires signing in so it can save your queries and for the sharing features.

CHAPTER 9

# Visual Data Views

TrainArrivalbyYear-Quarter

MostTendingStations2014

TrainArrival-ByStationYear

DelaysRateByDayHour

StationsTainAccuracy2015-v1

StationsTainAccuracy2015-v2

CHAPTER 10

# Code

https://github.com/hasadna/OpenTrainApp

# CHAPTER 11

## API for Android

Get all networks (GET): http://gtfs.otrain.org/api/data/bssids

Add new network (POST): http://gtfs.otrain.org/api/data/bssids/add/

```
{
  bssid: "ab:cd:ef:gh:ij:kl
  name: "Hashalom"
  stop_id: "37350"
}
```

Admin interface for manual update: http://gtfs.otrain.org/admin/

Stop list: http://gtfs.otrain.org/api/gtfs/stops/?format=json

Today's gtfs trips: http://gtfs.otrain.org/api/gtfs/trips/date/today/?format=json

Specific day's gtfs trips: http://gtfs.otrain.org/api/gtfs/trips/date/2015-09-10/?format=json

CHAPTER 12

# Updating the documentation

## 12.1 Simple instructions

Simply commit and push the code to github.

You can actually edit and preview changes directly in github!

The changes will be uploaded to readthedocs automatically through webHooks.

## 12.2 Difficult instructions

If you want to see the resulting html before you commit (usually you don't need to):

1. Run:

```
pip install sphinx
```

2. Make the updates you want to the .rst files, and then run:

```
make html
```

3. Open the index.html file in the browser to see the result.