Dashboard Guide

Dasboard Local Testing

1. Requirements: Docker
2. Open Terminal and switch to /Dashboard directory. From there you can directly build the Docker Container (All Dashboard sourcecode is inside the /files directory)
3. Build the docker container with the command: (The point at the end belongs to the command)
   `docker build -t racoon_dasbhoard .`
   This can take some minutes since the whole Python environment and all required pip packages will be downloaded.
4. Start the docker dashboard container with the command
   `docker run -p 8000:8000 -i -t racoon_dashboard`
   -p XXXX:YYYY means that port YYYY of the docker container will be mapped to port XXXX of the host PC
5. Open dashboard in browser:
   a. Startgpage: http://127.0.0.1:8000
   b. Admin View: http://127.0.0.1:8000/admin (Login Credentials: User: *admin* Password: *admin*)
   c. API View: http://127.0.0.1:8000/api (Attention: The API sites for DailyData, MeasureData, and PlotData may take some time to load since they show all available data on one page)

Notes:

- The Docker Container Dashboard is not persistent. This means, if the container is restarted, all changes are lost.
- All persistent data of the container are saved in *Dashboard/files/db.sqlite3* which is the database of the dashboard that is put into the container during build. If you change data there before building the container in step 2, this data will later be available in the container.

Database Architecture:

The following tables are used in the database. They are automatically build according to the definitions in */base/models.py*

- *Locations*: This table contains an entry for each location that will be displayed on the dashboard
  - *location_id*: A unique identifier for the location. Plan is to use the unique JIP-ID from each local node for this id. Will be needed for posting new data to the dashboard
  - *latitude*: Latitude coordinate of the location
  - *longitue*: Longitude coordinate of the location
  - *name_de*: German location name
  - *name_en*: English location name
  - *description_de*: A short german description about the location that will be displayed in the POI summary right below the location name.
  - *description_en*: The short description in english

- *Measure*: This table contains all measures that are shown on the dashboard and can be received via POST Requests
  - *id*: Unique identifier of the location. This will be needed when posting new data to the dashboard
  - *public_visible*: Boolean that defines if this measure should be visible for everyone and also users who are not logged in. If it is False, only logged in users with access to the corresponding location can see this measure.
  - *is_main*: Boolean that defines if the measure is considered a main measure. Main measures are the ones that can be chosen on the settings menu of the map view and can so be displayed on the map. Non-main-measures cannot be displayed as dot color / size on the map. However you can view their values by looking into the POI detail windows or by switching to the data view

    - *is_color_default*: Boolean that should only be True for one measure. This measure will be the default one that represents the color of the map POIs when the site is opened
    - *is_size_default*: Boolean that should only be True for one measure. Basically the same as „is_color_default“, but encoded this measure as the default POI sizes
    - *is_open_ended*: Boolean that should be True if there is no upper limit for this measure (e.g. continuously growing numbers). If this is False, the lower and upper bounds will be defined by the lowest and highest occurring values of this measure over all available entries
    - *name_de*: German name of the measure
    - *name_en*: English name of the measure
    - *description_de*: German description of the measure. Will be shown if the user hovers the small „i“ in the POI summary window or in the measure selection menu
    - *description_en*: English description of the measure
    - *lower_bound*: Value for the lowest possible measure value that can occur. Will only be respected if „is_open_ended“ is False
    - *upper_bound*: Value for the highest possible measure value that can occur. Will only be respected if „is_open_ended“ is False

- *DailyData*: This table maps the connection between locations and a specific date.
    - *location*: ForeignKey to a Locations table entry (defined by location_id)
    - *date*: the date of this entry

- *MeasureData*: This table builds the 1:n relation between a location-date entry and multiple measure values (ONE dailyData can have N measures with corresponding values):
    - *daily_data*: ForeignKey to a DailyData table entry (defined by location_id – date pair)
    - *measure*: Foreign Key to a Measure table entry (defined by measure_id)
    - *value*: The value for the given Measure – DailyData combination

- *PlotData*: This table builds the 1:n relation between a location-date entry and an arbitrary number of plots (ONE dailyData can have N plots). It’s a bit simpler than MeasureData since the plots don’t have to be predefined like the measure values that refer to the Measures table.
    - *plot_id*: Automatically generated id for the plot
    - *daily_data*: Foreign Key to a DailyData table entry
    - *public_visible*: Boolean that is True if this plot should be visible for all users. If False, only logged in users that have full access to the plot’s location can see it.
    - *name_de*: German name of the plot
    - *name_en*: English name of the plot
    - *plot_data*: A JSON object that describes the plot. All plots are Apache Echarts plots and the JSON represents the plot options. Here you can find many examples for these plots and their option JSONs: https://echarts.apache.org/examples/en/index.html

- *RacoonPermissions*: This is a special database that has no connection to the actual dashboard data. It is part of the user management and adds some additional permission options. You don’t have to change anything in this table. Users can be fully managed through the Dashboard’s admin view where new accounts can be created and permissions can be set.
    - *user*: 1:1 relation field to user
    - *visible_locations*: Many:Many field where multiple locations can be chosen where a given user can access all public and private (where the is_public attribute is False) measures.
    - *all_locations_visible*: Convenience option. Instead of selecting all locations in „visible_locations“, setting this to True will automatically grant access to all locations
    - *api_post_permission*: Has to be True if this user should be able to send HTTP Post Requests to the dashboard (Besides this the user also have to have an API token. More details about this are given in the next section)

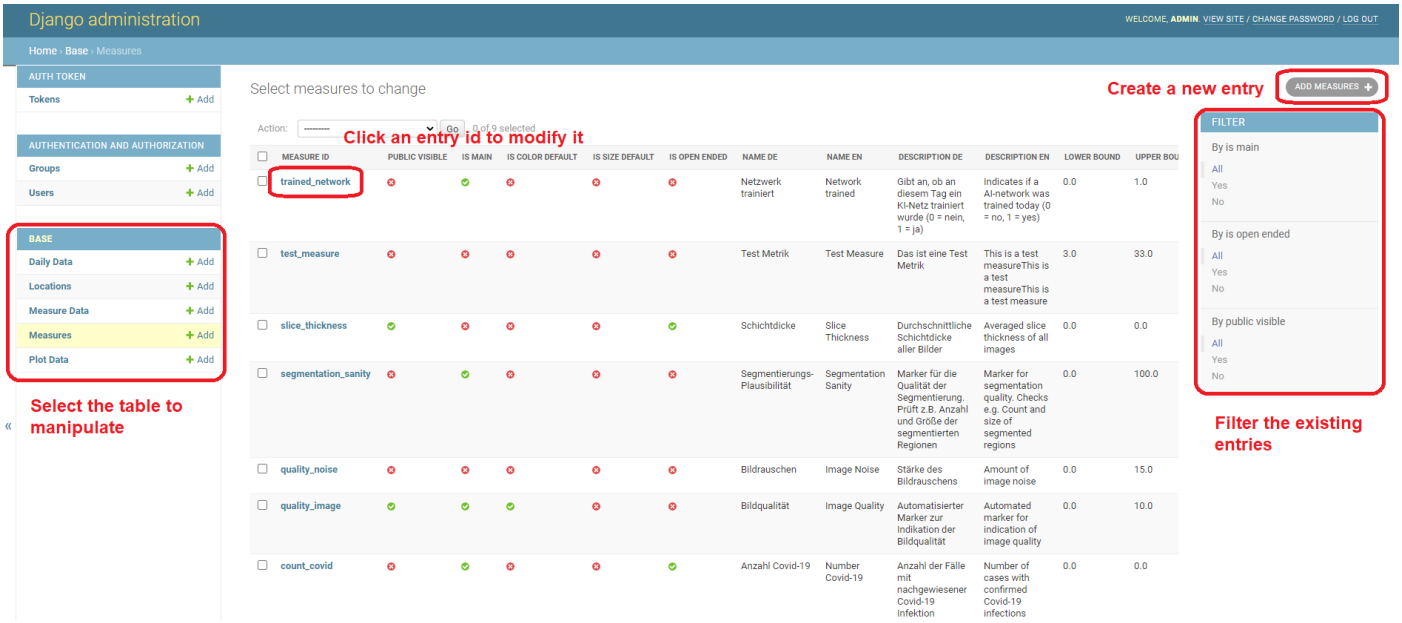
Manipulating Data

There are four ways of changing data in the dashboard:

- In the admin view
- Via HTTP-Request
- Manually in the database
- During Initial Deployment

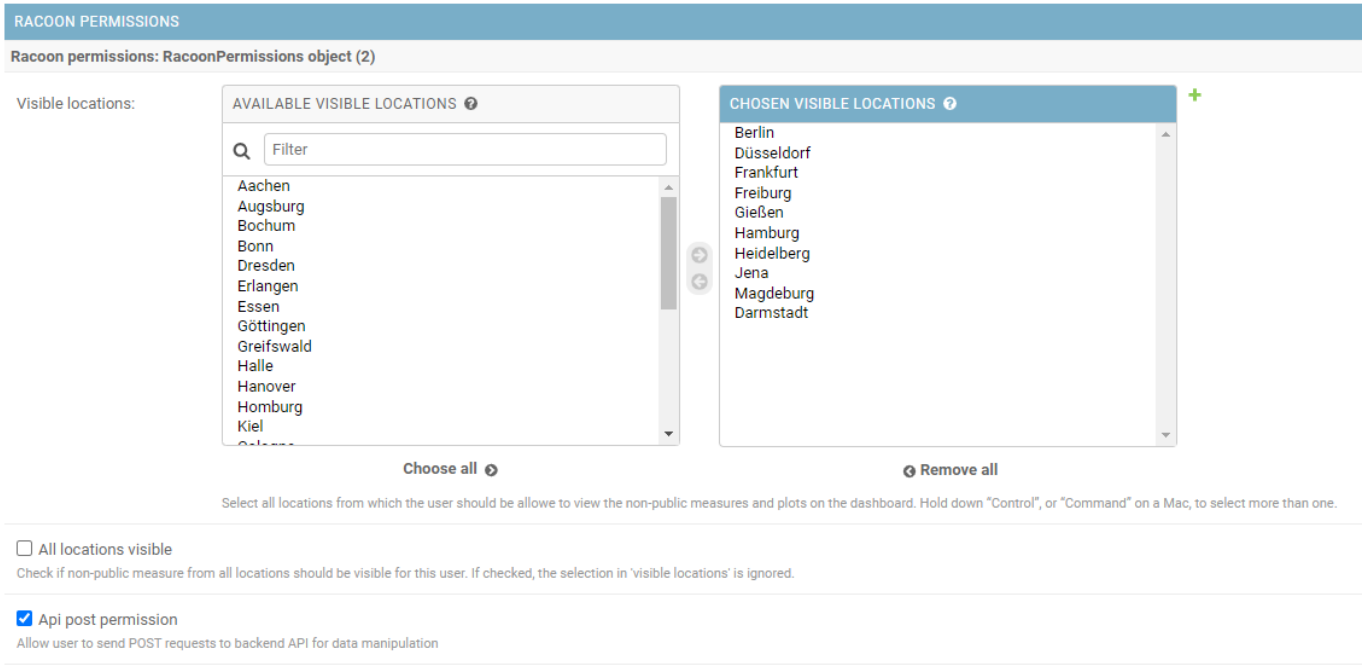
Admin View:

Using the admin view for manipulating dashboard data is the most straight-forward way. You can access the Admin view via *<dashboard-url>/admin* . Here you can simply select that tables, filter entries, alter existing ones or create new data rows



Here you can also create new users by selecting the Users table under Authentication and Authorization. New users require an username and a password.

To modify the permissions of existing user, you can click the username from the overview table. Then scroll down to find the racoon permissions. Here you can select locations from which the user should be allowed to view the *is_public = False* measure:



HTTP-Request

Basics

This is the main method to update data in the dashboard. It will be used primarily by JIP workflows to push new aggregated data entries or update existing ones.

There are three kinds of HTTP Requests available:

- *GET* Requests: These are used to obtain data. They are used by the dashboard's frontend to load the data into its different views.
- *POST* Requests: Create new data instances
- *PATCH* Requests: Update existing measure entries.

In order to use POST and PATCH Requests, an API Token is necessary. This token can be generated inside the Admin view. Go to the *Tokens* table under *Auth Token* and click on *Add Token* in the top right corner. Then you are prompted to select the user account for whom the token should be generated. Afterwards the token will be shown as Key in the Tokens overview.

Note that the you also have to grant the API post permission to the user who will be sending the requests. This can be done by selecting the a user from the *Users* table where you can scroll to the bottom and tick the box *Api post permission* .

With these settings in place you can send out the requests. The basic structure has to be:

- Headers:
  - Authorization: `Token <API_TOKEN>`
    Note that the word Token before the actual Token is mandatory
  - Content Type: `application/json`
- Body: JSON data for the corresponding request

Endpoints & Examples

- Adding new locations:
  - Endpoint: *<dashboard-url>/api/locations/*
  - Request Type: POST
  - Example Request Body:

```
{
        "location_id": "aug",
        "latitude": "48.36680410",
        "longitude": "10.89869710",
        "name_de": "Augsburg",
        "name_en": "Augsburg",
        "description_de": "Augsburg ist eine der ältesten Städte Deutschlands.",
        "description_en": "Augsburg is one of Germany's oldest cities."
}
```

- Adding new measures:
  - Endpoint: *<dashboard-url>/api/measures/*
  - Request Type: POST
  - Example Request Body:

```
{
    "measure_id": "count_all",
    "public_visible": true,
    "is_main": true,
    "is_color_default": false,
    "is_size_default": true,
    "is_open_ended": true,
```

```
-       "lower_bound": 0.0,
-       "upper_bound": 0.0,
-       "name_de": "Anzahl Gesamt",
-       "name_en": "Number Total",
-       "description_de": "Die Zahl aller Fälle",
-       "description_en": "Number of cases"
-   }
```

- Adding new data:
    - Endpoint: *<dashboard-url>/api/data/*
    - Request Type:
        - POST: Create new instance
        - PATCH: Update or Create new instance (Note that existing values will be overwritten)
    - Example Request Body:

```
-   {
-       "location": "cha",
-       "date": "2020-01-21",
-       "measureData":[
-           {
-               "measure":"test_id",
-               "value":"0.5"
-           },
-           {
-               "measure":"quality_noise",
-               "value":"10"
-           }
-       ],
-       "plotData":[
-           {
-               "name_de": "Plot Name deutsch",
-               "name_en": "Plot name english",
-               "public_visible": true,
-               "plot_data": {
-                   xAxis: {
-                       type: 'value',
-                   },
-                   yAxis: {
-                       type: 'value'
-                   },
-                   series: [{
-                       data: [150, 230, 224, 218, 135, 147, 260],
-                       type: 'line'
-                   }]
-               }
-           }
-       ]
-   }
```

    - Note that both attributes, *measureData* and *plotData* may be an empty list []. In these cases only the daily data entry will be created that can later be updated with a PATCH request.

Manually changing data in the database

This possibility of altering data of the dashboard depends on the currently utilized database type. We differ two different setups, a local one in DEV mode, and a production deployment which utilize different database types:

- SQLite: This database is used for the DEV version of the dashboard. It is stored locally in the dashboard directory and can be simply altered by opening it with a SQLite Explorer.
- MySQL: This database is used in the actual production environment when the dashboard is deployed. The database credentials are stored inside the dashboard settings which can be found in */racoon/settings.py* under *DATABASES*

Change Data during initial deployment

The last option for adding dashboard data is during the initial deployment (also locally). As previously stated, Django creates migration files that are automatically generated to build the database according to the models defined in */base/models.py*. These migrations can be found in */base/migration/…* . The *0001_initial.py* migration must not be altered and changes will result in errors when the dashboard is run. However, the initial data of the dashboard is added in *0002_initial_data.py* and may be altered as you want or even deleted when there should not be any initial data after deployment. The current version adds all RACOON locations (Note that the ids must be changed later on to the corresponding JIP unique identifiers), some test measures, 90 days of random data for these test measures, and a random selection out of five different plots to each DailyDate entry. Feel free to make changes to the code or try out other data.

LIVE Deployment (Sourcecode Location: /Dashboard/files)

Here are all steps that are required to prepare the dashboard for LIVE deployment:

1. Setup an empty MySQL database
2. Prepare the dashboard configuration in */racoon/settings.py*
    a. Set the value of ENVIRONMENT to PROD. (*DEV* is only meant for testing)
    This will ensure that DEBUG mode is turned off
    b. Configure the database settings of the dict DATABASES
    Either provide the database credentials as environmental variables:
        - RDS_HOSTNAME: MySql DB Hostname
        - RDS_PORT: MySql DB Port
        - RDS_DB_NAME: MySql database name
        - RDS_USERNAME: MySql username
        - RDS_PASSWORD: MySql password

    Or remove the part with the environmental variables and set the attributes manually
    c. Set the value of USE_LOCKDOWN to False. Lockdown is an addon that locks the whole website behind a password input field. If it is activated, users have to login with a password to view the dashboard. Note that POST and GET requests will fail since they cannot pass this lockdown page
3. Prepare the initial data migrations in */base/migration/…*
    a. *0001_initial.py* – Do not change anything here!
    b. *0002_initial_data.py*
        i. Remove all code that creates the fake data
        ii. Replace the placeholder measures with the real measures
        iii. Check the locations and remove / add wrong entries
        iv. Replace all location ids with the JIP node ids from the local JIP instances. If these are not known, it can be changed later on (e.g. in the admin view)
4. Prepare the Python environment.
    a. Install Python (development version was Python 3.6.7)
    b. Install the requirements with `pip install -r requirements.txt`
5. Execute the command `python manage.py migrate` to initialize the database (You only have to run this command once)
6. Start the webserver: `python manage.py runserver`

7. Create an Admin user: `python manage.py createsuperuser`
   Then follow the descriptions in the command line and enter an username and password