Master Facility List Kenya Documentation

Release 0.0.1a3

developers@savannahinformatics.com

Aug 05, 2017

Contents

1 Developers guide **3**

1.1 Installing for evaluation . . . 3
1.2 Installing for development . . . 4
1.3 Installing for production . . . 5
1.4 The big picture . . . 7
1.5 Authentication and authorization . . . 8
1.6 Using the API - basic principles . . . 16
1.7 The API sandbox . . . 23
1.8 Metadata resources . . . 23
1.9 The service catalog . . . 30
1.10 Facilities . . . 34
1.11 Facility types . . . 50
1.12 Facility Upgrades and Downgrades . . . 52
1.13 Community Health Units . . . 55
1.14 Regulation . . . 58
1.15 GIS Support . . . 68
1.16 Workflow . . . 72
1.17 Contributors' code of conduct . . . 73
1.18 Regulator Synchronization . . . 73

This is documentation for the API server for the second generation Kenyan Ministry of Health Master Facility List (MFL). The MFL system's "home" is at http://ehealth.or.ke/facilities/ . This documentation is aimed at developers (both MFL developers and those developing third party systems that use the MFL API) and system administrators.

There is a downloadable PDF version of this documentation, a mobile friendly EPUB version and a downloadable HTML version.

CHAPTER 1

Developers guide

Installing for evaluation

In this scenario, you do not plan to make any changes to the MFL API server but you need to have a local copy against which you can test a new API client or a new third party integration.

We recommend that you use Vagrant and Virtualbox to create a test server for yourself.

If you are an expert Vagrant user, you can substitute Virtualbox with VMWare Desktop / Player, HyperV etc. You'll have an easier time if you are on a `_nix` e.g Ubuntu or OS X.

Deployment Assumptions

The deployment scripts will fail unless the following are true:

- you are on a vagrant supported OS (so far Ubuntu 14.04LTS has been tested)
- you have run `ssh-keygen` and have a public key at `$HOME/.ssh/id_rsa.pub`

Vagrant

Before installation, you will need to have the vagrant-env plugin. The installation is as simple as running

```
vagrant plugin install vagrant-env
```

Ansible is used to provision the vagrant box. An understanding of ansible is recommended though not required.

Installation

1. Ensure vagrant is installed
2. Create a python virtual environment and activate the created virtual environment.

3. Install `ansible` in the virtual environment.
4. Set the following environment variables:
    - `DATABASE_NAME` the name of the database to user
    - `DATABASE_USER` the database user to use
    - `DATABASE_PASSWORD` the database password to use
5. Run `vagrant up`. It shall download and setup everything in the virtual machine.
6. The system is ready to use

Installing for development

You'll have an easier time if you are on a current Ubuntu. On Ubuntu, the key dependencies can be installed with:

```
sudo apt-get install postgresql binutils postgis gdal-bin libproj-dev
libgeoip1 graphviz libgraphviz-dev
```

To build lxml on Debian 8 you have to install:

```
sudo apt-get install libxml2-dev libxslt-dev
```

You may need to install distribution specific packages e.g on Ubuntu 14.04 with the default PosgreSQL 9.3:

```
sudo apt-get install postgresql-9.3-postgis-2.1
```

In order to build some of the Python dependencies in the *virtualenv*, some libraries will need to be in place. Again, if you are on a recent Ubuntu, you can get them at once with:

Note: This project has been tested with Python2. It may work with Python3 but it has not been tested

---

```
sudo apt-get build-dep python-shapely python-numpy cython python-psycopg2
libxml2-dev libxslt1-dev libffi-dev
```

Note: You must ensure that ElasticSearch is running. In a typical Ubuntu install (from the *.deb* supplied by ElasticSearch), the search server is not started by default.

Getting started

A: Running the system from source code

1. Create a virtualenv
2. Activate the created vitualenv and run `pip install -r requirements.txt`
3. **Run the following commands sequentially:**
    - **fab setup_db** This drops the database if it exists, creates the database and runs migrations.
    - **fab load_demo_data** This will load sample test data for the API if the project **DEBUG** attribute in settings is set to *True*.

- **fab recreate_search_index** Creates an Elasticsearch index. Before running this command ensure that Elasticsearch is up and running. This command causes the data that has been loaded in the database to be indexed in ElasticSearch.

Note: At times during development one may want to retain the database. To do so, call `fab load_demo_data` and `fab recreate_search_index`.

Also one may want to recreate the database. Calling `fab setup_db` drops the database, creates it again and runs migrations. After this one may proceed to load the data

and create the search index as desired.

B: Installing the system Activate the virtualenv and run `python setup.py install` while in the project folder.

Installing for production

This server has been developed and tested on Ubuntu Linux (any Ubuntu that is currently "in support" will do). It should be *trivial* to get it working on any `*NIX` (including OS X).

Kindly note that this restriction applies to the servers only, and not to any of the API clients e.g browsers and third party systems. Clients can run on any modern operating system.

We supply an Ansible playbook that automates this entire process.

Setting up the environment

This server is built as a Twelve-Factor App. For that reason, the key configuration parameters are stored in the environment - set up directly in the operating system as environment variables or as a *.env* file in the application's root folder.

The `.env` file holds confidential configuration information. For that reason, it is not tracked in version control (version control has an example `.env` whose values should **not** be used in production).

A proper `.env` file should set the following values up:

```
SECRET_KEY=pleasechangetoanewlygeneratedsecretkey
DEBUG=off  # NEVER run with Debug=True in production

# Use real email settings here e.g from Amazon SES
EMAIL_HOST=''
EMAIL_HOST_USER=''
EMAIL_HOST_PASSWORD=''


# Here because the original user was too lazy to write ruby code for the VagrantFile
DATABASE_USER=mfl  # Change this
DATABASE_PASSWORD=mfl  # **CHANGE** this, no matter how lazy you feel
DATABASE_NAME=mfl  # Change this

# Make sure you change this in lockstep with the three DATABASE_* vars above
DATABASE_URL='postgres://mfl:mfl@localhost:5432/mfl'


# Location where the administration frontend is running
```

```
FRONTEND_URL='http://localhost:8062'
DEBUG=False
REALTIME_INDEX=True  # ** set to true to update the search index in realtime**
HTTPS_ENABLED=True # ** Set to true if HTTPS will be used
AWS_ACCESS_KEY_ID=<AWS access key>
AWS_SECRET_ACCESS_KEY=<AWS secret key>
AWS_STORAGE_BUCKET_NAME=<AWS bucket name>
STORAGE_BACKEND=<storage backend e.g storages.backends.s3boto.S3BotoStorage>
```

Warning: Please make sure that you have set up secure values.

You will need to save a copy of the `.env` at a secure location (not in the code repository). If you loose the `.env` / forget the values, you could lose the ability to maintain the deployed production system.

The pre-deploy checklist

You **MUST** work your way through the Django deployment checklist.

Configuring the ansible inventory

There is an `inventory` file in the `playbooks` folder. This file should be edited to have a line for each server that is managed by Ansible.

The following is an example:

```
azure_test_server              ansible_ssh_host=mfl.slade360.co.ke     ansible_ssh_
↪port=22     ansible_ssh_user=azureuser     ansible_ssh_private_key_file=/home/
↪ngurenyaga/.ssh/id_rsa
```

The template breaks down roughly to this:

```
<a descriptive name we choose for the server>
ansible_ssh_host=<an IP address or host name>
ansible_ssh_port=<the port over which the SSL daemon is listening on the remote␣
↪machine>
ansible_ssh_user=<the username to log in with on the remote machine>
ansible_ssh_private_key_file=<a path to a local SSH private key>
```

Warning: The SSH private key must be kept private.

In the site.yml file ensure that the relevant variables are updated e.g

```
mfl_public_web_version: "0.0.1a13" //set to the version of public website that should␣
↪be deployed
mfl_admin_web_version: "0.0.1a21" //set version of the administration site to be␣
↪deployed
has_ssl: true // set this to true if the site should run on HTTPs
cert_file: "" // Give the location of the HTTPS certificate file
key_file: "" // Give the location of the HTTPS key file
public_web_server_name: "public.test_domain.com" //The public website URL
admin_web_server_name: "public.test_domain.com" //The administration website URL
```

```
load_demo_data: false , //set to true if demonstration data needs to be loaded
warm_cache: false, // set this to true if the cache needs to be refreshed
server_url: "https://testdomain.com, //THe API-server URL
username: <Public user username>
password: <public password>,
client_id: <OAUTH client id for the public user >,
client_secret:<OAUTH client secret for the public user>
```

Once all the deployment attributes have been set, **cd** into the **playbooks** folder and run the command below:

```
ansible-playbook site.yml
```

It deploys the API server, the public website and the administration website and they should be available from the URLs provided in the site.yml file once the command has finished executing.

Warning: If you are working off a recent Ubuntu Linux on your laptop, you should comment out `SendEnv LANG LC_*` in `/etc/ssh/ssh_config`.

The forwarding of language environment variables from the local computer is known to cause mischief on the remote server.

Warning: This server should only be run on a non-threaded server e.g `gunicorn` in the standard multi-process configuration.

This is because the geographic features rely on `GDAL`, which is not thread safe.

The big picture

The Master Facilities List is one of the building blocks of the Kenyan National Health Information System. The second edition of the MFL is focused on interoperability, standardization and unification.

Interoperability

This system adopts an **API First** approach, as explained in the *Using the API - basic principles* chapter.

The authors have gone to great lengths to make it easy for other systems - with the correct authorization - to read and write MFL data.

Standardization

The MFL's core mission includes the standardization of facility codes. In this edition, the core mission has been expanded to include the standardization of service codes. You can read more about that in the *The service catalog* chapter.

Unification

The first generation of the Master Facilities List (and its "satellites") had five semi-independent systems: public and administration systems for the "core" MFL, a mirror of those two for the Master Community Units List and a

regulators interface.

This release unifies them all under a single API. That API is client agnostic - the client could be a web or mobile application, another system or even a reporting tool.

Note: A future release of this system could standardize more things e.g practitioner codes.

Authentication and authorization

Authentication is the process of associating an API request with a specific user, while authorization determines if the user has permission to perform the requested operation.

Authentication

A system like this has to consider the needs of programmatic clients (like integrations into other systems) and the needs of "actual users" (in this case people logged in to the web interfaces).

The MFL API server supports both session (cookie based) and OAuth 2 (token based) authentication. For both approaches, the production API server **must be run over HTTPS**.

Session Authentication

Logging in

`POST` the credentials to `/api/rest-auth/login/`. The payload should be similar to the example below:

```
{
    "username": "hakunaruhusa@mfltest.slade360.co.ke",
    "password": "hakunaruhusa"
}
```

A successful login will have a `HTTP 200 OK` response. The response payload will have a single `key` parameter: a Django Rest Framework TokenAuthentication key. For example:

```
{
    "key": "f9a978cd00e9dc0ebfe97d633d98bde4b35f9279"
}
```

Note: Please note that the `username` is actually an email address.

Note: We discourage the use of token authentication. Kindly see the section on OAuth2 below.

Logging out

Send an empty (no payload) `POST` to `/api/rest-auth/logout/`.

A successful logout will get back a `HTTP 200 OK` response, and a success message similar to the one below:

```
{
    "success": "Successfully logged out."
}
```

Getting user details after login

After a user is logged in, a typical client (such as a web application) will need to get additional information about the user. This additional information includes permissions.

If the user is logged in, a `GET` to `/api/rest-auth/user/` will get back a `HTTP 200 OK` response and a user details payload similar to this example:

```
{
    "id": 3,
    "short_name": "Serikali",
    "full_name": "Serikali Kuu ",
    "all_permissions": [
        "common.add_town",
        "oauth2_provider.change_accesstoken",
        "mfl_gis.delete_wardboundary",
        "auth.add_permission",
        "chul.change_approvalstatus",
        "facilities.delete_facilitytype",
        // a long list of permissions; truncated for brevity
    ],
    "user_permissions": [],
    "groups": [],
    "last_login": "2015-05-04T16:33:36.085065Z",
    "is_superuser": true,
    "email": "serikalikuu@mfltest.slade360.co.ke",
    "first_name": "Serikali",
    "last_name": "Kuu",
    "other_names": "",
    "username": "serikalikuu",
    "is_staff": true,
    "is_active": true,
    "date_joined": "2015-05-03T02:39:03.440962Z",
    "is_national": true,
    "requires_password_change": false
}
```

If the user is not logged in, the return message will be a `HTTP 403 FORBIDDEN` with the following message:

```
{
    "detail": "Authentication credentials were not provided."
}
```

Note: If a user needs to change their password e.g because it was created by an admin and must be changed on first login, the `requires_password_change` boolean property will be set to `true`.

Every well behaved web client should observe this property and implement the appropriate "roadblock".

OAuth2 Authentication

You can learn all that you need to know about OAuth2 by reading rfc6749.

A simple OAuth2 workflow

If you are in too much of a hurry to read all that, here is what you should do:

Registering a new "application"

You should know the user ID of the user that you'd like to register an application for. You can obtain that ID from the user details API described above or from `/api/users/`.

You need to know the `authorization_grant_type` that you'd like for the new application. For the example below, we will use `password`. If you do not know what to choose, read rfc6749 .

The next decision is the choice of `client_type`. For the example below, we will use `confidential`. As always - consult rfc6749 for more context.

`POST` to `/api/users/applications/` a payload similar to this example:

```
{
    "client_type": "confidential",
    "authorization_grant_type": "password",
    "name": "Demo / Docs Application",
    "user": 3
}
```

A successful `POST` will get back a `HTTP 201 CREATED` response, and a representation of the new application. This example request got back this representation:

```
{
    "id": 1,
    "client_id": "<redacted>",
    "redirect_uris": "",
    "client_type": "confidential",
    "authorization_grant_type": "password",
    "client_secret": "<redacted>",
    "name": "Demo / Docs Application",
    "skip_authorization": false,
    "user": 3
}
```

Note:

- The *client_id* and *client_secret* fields were automatically assigned.
- The *skip_authorization* and *redirect_urls* fields have default values.
- A single user can be associated with multiple applications.

Authenticating using OAuth2 tokens

First, obtain an access token by `POST` ing the user's credentials to `/o/token/`. For example:

```
curl -X POST -d "grant_type=password&username=serikalikuu@mfltest.slade360.co.ke&
↪password=serikalikuu" http://
↪sfzgvKKVpLxyHn3EbZrepehJnLn1r0OOFnuqBNy7:7SMXKum5CJVWABxIitwszES3Kls5RTBzYzJDI5jdvgPcw0vSjP5pnlYHfA
↪o/token/
```

Which breaks down as:

```
curl -X POST -d grant_type=<grant_type>&username=<email>&password=<password>" http://
↪<client_id>:<client_secret>@<host>:<por>/o/token/
```

If you authenticate successfully, the reply from the server will be a *JSON* payload that has the issued access token, the refresh token, the access token type, expiry and scope. For example:

```
{
    "access_token": "fKDvh2fFLR1iFPuB26RUEalbjYO4rx",
    "token_type": "Bearer",
    "expires_in": 36000,
    "refresh_token": "jLwpCh3WbOXBeb01XMeZR5AQYedkj1",
    "scope": "read write"
}
```

Pick the `access_token` and send it in an `Authorization: Bearer` header e.g

```
curl -H "Authorization: Bearer ziBLqoXwVEA8lW9yEmE260AZ4lCJHq" http://localhost:8000/
↪api/common/counties/
```

Authorization

This server's Role Based Access Control setup is based on the Django framework permissions and authorization system.

Understanding the role based access control setup

The user details API endpoint (explained above) returns the logged in user's permissions.

A user's permissions come from three "sources":

- the permissions assigned to the group (role) that the user belongs to
- the permissions assigned directly to the user
- the `is_superuser` boolean flag; a user who is a "superuser" automatically gets all permissions

The MFL API server has an additional "layer" of authorization: whether a user is a "national user" or a "county user". In certain list endpoints (chiefly those that deal directly with facilities), a "county" user will have their results limited to facilities that are located in their county.

Note: This API server does not support "true" unauthenticated read-only access For the public site, OAuth2 credentials (that correspond to a role with limited access) will be used.

Note: From the point of view of the MFL API, regulator systems are just one more set of API clients.

Users and counties

In 2010, Kenya got a new constitution. One of the major changes was the establishment of a devolved system of government.

The second generation MFL API (this server) is designed for the era of devolution. In this system, facility record management should occur at the county level.

The separation of privileges between data entry staff ("makers") and those responsible for approval ("checkers") can be modelled easily using the role based access control setup described above.

The only additional need is the need to link county level users to counties, and use that information to limit their access. This has been achieved by adding an `is_national` boolean flag to the custom user model and adding a resource that links users to counties. The example user resource below represents a non-national (county) user (note the `is_national` field):

```
{
    "id": 4,
    "short_name": "Serikali",
    "full_name": "Serikali Ndogo ",
    "all_permissions": [
        "common.add_town",
        // many more permissions
    ],
    "user_permissions": [],
    "groups": [],
    "last_login": null,
    "is_superuser": true,
    "email": "serikalindogo@mfltest.slade360.co.ke",
    "first_name": "Serikali",
    "last_name": "Ndogo",
    "other_names": "",
    "username": "serikalindogo",
    "is_staff": true,
    "is_active": true,
    "date_joined": "2015-05-03T02:39:03.443301Z",
    "is_national": false
}
```

In order to link a user to a county, you need to have two pieces of information:

- the user's `id`
- the county's `id` - easily obtained from `/api/common/counties/`

With these two pieces of information in place, `POST` to `/api/common/user_counties/` a payload similar to this example:

```
{
    "user": 4,
    "county": "d5f54838-8743-4774-a866-75d7744a9814"
}
```

A successful operation will get back a `HTTP 201 CREATED` response and a representation of the newly created resource. For example:

```
{
    "id": "073d8bfa-2a86-4f9a-9cbe-0b8ac6780c3a",
    "created": "2015-05-04T17:44:56.441006Z",
    "updated": "2015-05-04T17:44:56.441027Z",
```

```
    "deleted": false,
    "active": true,
    "created_by": 3,
    "updated_by": 3,
    "user": 4,
    "county": "d5f54838-8743-4774-a866-75d7744a9814"
}
```

The filtering of results by county is transparent (the API client does not need to do anything).

Note: A user can only have one active link to a county at any particular time. Any attempt to link a user to more than one county at a time will get a validation error.

If you'd like to change the county that a user is linked to, you will need to first inactivate the existing record (`PATCH` it and set `active` to `false`).

In order to determine the role that a user is currently linked to, issue a `GET` similar to `/api/common/user_counties/?user=4&active=true`. In this example, `4` is the user's `id`.

Setting up users, permissions and groups

Permissions

API clients should treat permissions as "fixed" builtins. The server does not implement any endpoint that can be used to add, edit or remove a permission.

The available permissions can be listed by issuing a `GET` to `/api/users/permissions/`. The results will look like this:

```
{
    "count": 216,
    "next": "http://localhost:8000/api/users/permissions/?page=2",
    "previous": null,
    "results": [
        {
            "id": 61,
            "name": "Can add email address",
            "codename": "add_emailaddress",
            "content_type": 21
        },
        {
            "id": 62,
            "name": "Can change email address",
            "codename": "change_emailaddress",
            "content_type": 21
        },
        {
            "id": 63,
            "name": "Can delete email address",
            "codename": "delete_emailaddress",
            "content_type": 21
        },
        // truncated for brevity
    ]
}
```

Note: The *content_type* keys in the example above originate from Django's contenttypes framework. For an API consumer, they are an implementation detail / curiosity; API clients will nto need to know more about them.

Groups

The API server provides APIs that can be used to create roles, alter existing roles and retire roles.

Existing roles (groups) can be listed by issuing a `GET` to `/api/users/groups/`.

Creating a new role

`POST` to `/api/users/groups/` a payload that similar to the one below:

```
{
    "name": "Documentation Example Group",
    "permissions": [
        {
            "id": 61,
            "name": "Can add email address",
            "codename": "add_emailaddress"
        },
        {
            "id": 62,
            "name": "Can change email address",
            "codename": "change_emailaddress"
        }
    ]
}
```

A successful operation will get back a `HTTP 201 CREATED` status.

Note: You must supply both a `name` and `permissions`.

Updating an existing role

`PUT` or `PATCH` to a group **detail URL** e.g `/api/users/groups/1/`.

For example, to take away from the example role the "Can change email address" permission, the following `PATCH` request should be sent:

```
{
    "permissions": [
        {
            "id": 61,
            "name": "Can add email address",
            "codename": "add_emailaddress"
        }
    ]
}
```

A similar approach will be followed to add permissions.

A successful operation will get back a `HTTP 200 OK` status.

Note: **Permissions will always be overwritten** when you perform an update.

User management

User registration (sign up)

`POST` to `/api/rest-auth/registration/` a payload similar to this example:

```
{
    "username": "likeforreal",
    "email": "likeforreal@yodawg.dawg",
    "password1": "most_secure_password_in_the_world_like_for_real",
    "password2": "most_secure_password_in_the_world_like_for_real"
}
```

A successful operation will get back a `HTTP 201 CREATED` response and a representation of the new user. For example:

```
HTTP 201 CREATED
Content-Type: application/json
Vary: Accept
Allow: POST, OPTIONS, HEAD

{
    "id": 9,
    "short_name": "",
    "full_name": "  ",
    "all_permissions": [],
    "user_permissions": [],
    "groups": [],
    "last_login": "2015-05-05T09:12:01.888514Z",
    "is_superuser": false,
    "email": "likeforreal1@yodawg.dawg",
    "first_name": "",
    "last_name": "",
    "other_names": "",
    "username": "likeforreal1",
    "is_staff": false,
    "is_active": true,
    "date_joined": "2015-05-05T09:12:01.790167Z",
    "is_national": false
}
```

Note: This API server does not implement email address confirmation. A future release might implement that.

Note: The registration operation described above suffices, for public users.

The manner in which users should be linked to counties has already been discussed in the Authorization section.

Linking users to groups

In order to assign a user to a group, you will need to know the group ID (which you can obtain from `/api/groups/`).

`PATCH` an already existing user with a payload similar to this example:

```
{
    "groups": [
        {"id": 1, "name": "Documentation Example Group"}
    ]
}
```

In order to remove them from their assigned roles, `PATCH` with an empty `groups` list.

Note: This server does not support the direct assignment of permissions to users. That is deliberate.

Updating user details

Every writable attribute of a user record can be `PATCH``ed. For example, to inactivate or retire a user, ``PATCH` the user's ( detail ) record and set `is_active` to `false`.

For example: if the detail record for the user we registered above (`likeforreal`) is to be found at `/api/users/9/`, the user can be inactivated by `PATCH` ing `/api/users/9/` with:

```
{
    "active": false
}
```

Note: The same general approach can be used for any other flag e.g `is_superuser`.

Password changes

The password of the **logged in user** can be changed by making a `POST` to `/api/rest-auth/password/change/` a payload similar to this example:

```
{
    "old_password": "oldanddonewith",
    "new_password1": "newhotness",
    "new_password2": "newhotness"
}
```

Note: A future version of this server may add support for social authentication e.g login via Facebook, Twitter or Google accounts.

Using the API - basic principles

All the material here assumes that you already have access to an MFL test environment.

See *The API sandbox* and *Installing for evaluation* or *Installing for development* for information on how to get access to a test environment.

The MFL v2 project subscribes to the API First approach. It is **built to interoperate**. We "eat our own dog food" by insisting that the official user interfaces be just one more set of API clients, with no special privileges.

This guide is for the authors of client applications (applications that consume the RESTful web e.g. *The service catalog*). Those who would like to make changes to the MFL API server code itself should refer to the *Workflow* guide.

The MFL 2 API is "RESTish". We subscribe to the principles of REST but are not pedantic about it. It is built using the excellent Django REST Framework.

HTTP and HTTPS

All API actions are based on HTTP and its verbs e.g. `GET` and `POST`.

HTTP Verb	Description
HEAD	Used to **retrieve header information** about a resource
GET	Used to **retrieve a resource** and for any **read-only operation**
POST	Used to **create a resource** and sometimes to change it
PUT	Used to **mutate an existing resource**. We, however, encourage the use of `PATCH` instead of `PUT` whenever possible.
PATCH	Used to **edit** an already existing resource
DELETE	Used to **delete** an already existing resource

Production instances should always run over HTTPS.

Data Format

The MFL API server supports JSON for all API endpoints.

Some endpoints support CSV and Excel output. This will be indicated in the relevant sections of the documentation.

The preferred data format is JSON. **We strongly encourage you to use JSON** - you will find it to be more reliable, since it is the format used by the official front-ends and is therefore extensively tested.

In order to request a specific format, you will need to learn how to use content negotiation .

Content Negotiation using headers

Send the correct `Accept` header. For example:

To get json

curl -i -H "Accept: application/json" -H "Content-Type: application/json" http://localhost:8000/api/common/contacts/

To get csv

curl -i -H "Accept: application/csv" -H "Content-Type: application/csv" http://localhost:8000/api/common/contacts/

To get a resource in Microsoft Excel format

curl -i -H "Accept: application/xlsx" -H "Content-Type: application/vnd.openxmlformats-officedocument.spreadsheetml.sheet" http://localhost:8000/api/common/contacts/

Please note that the examples above do not factor in *Authentication and authorization*.

Content negotiation using query parameters

Append a `?format=<>` `GET` parameter. For example:

- to get JSON (the default), append `?format=json` to the URL
- to get CSV append `?format=csv` to the URL
- to get Excel, append `?format=excel` to the URL

Common fields

All MFL (Master Facilities List) resources have the following fields:

Field	Description
`id`	A UUID. This is the database record's primary key.
`created`	An ISO 8601 timestamp (UTC time zone) that indicates when the resource was created
`updated`	An ISO 8601 timestamp (UTC time zone) that shows when the last update occured
`active`	A boolean; will be set to `false` when the record is retired
`deleted`	A boolean; will be set to `true` when the record is removed. The API will in-fact not return deleted items by default.
`created`	The ID of the user that created the record. The user model is the only one with non UUID primary keys.
`updated`	The ID of the user that last updated the record.

The example listing below clearly shows the shared fields:


```
{
    "count": 5,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": "16f7593f-0a21-41b6-87f1-ef2c4ec7e029",
            "created": "2015-05-03T02:30:26.345994Z",
            "updated": "2015-05-03T02:30:26.346007Z",
            "deleted": false,
            "active": true,
            "name": "POSTAL",
            "description": null,
            "created_by": 1,
            "updated_by": 1
        },
        {
            "id": "f4eaf905-be91-4050-b154-600e31510306",
            "created": "2015-05-03T02:30:26.342216Z",
            "updated": "2015-05-03T02:30:26.342229Z",
            "deleted": false,
            "active": true,
            "name": "FAX",
            "description": null,
            "created_by": 1,
            "updated_by": 1
        },
        {
```

```
            "id": "f4e835d3-e6a4-4d2d-9d37-344a3da1bb0a",
            "created": "2015-05-03T02:30:26.338468Z",
            "updated": "2015-05-03T02:30:26.338481Z",
            "deleted": false,
            "active": true,
            "name": "LANDLINE",
            "description": null,
            "created_by": 1,
            "updated_by": 1
        },
        {
            "id": "68281bd2-d616-418d-ab01-616a225b643b",
            "created": "2015-05-03T02:30:26.334496Z",
            "updated": "2015-05-03T02:30:26.334510Z",
            "deleted": false,
            "active": true,
            "name": "MOBILE",
            "description": null,
            "created_by": 1,
            "updated_by": 1
        },
        {
            "id": "b2ce5bc9-0c73-4586-b5d2-e96c69b90b85",
            "created": "2015-05-03T02:30:26.328938Z",
            "updated": "2015-05-03T02:30:26.328956Z",
            "deleted": false,
            "active": true,
            "name": "EMAIL",
            "description": null,
            "created_by": 1,
            "updated_by": 1
        }
    ]
}
```


These fields are exposed via filters in most list endpoints. The examples below show those filters in use:

Filter	Example and examples
`updated_before`	Returns results where the date is less than or equal to the supplied timestamp. An example of a valid query is `GET /api/facilities/facilities/?updated_before=2014-05-06T10:36:45.112488Z`
`updated_after`	Returns results where the date is greater than or equal to the supplied timestamp. An example of a valid query is `GET /api/facilities/facilities/?updated_after=2014-05-06T10:36:45.112488Z`
`created_before`	Same as for `updated_before`, but operates on creation timestamps
`created_after`	Same as for `updated_after`, but operates on creation timestamps
`is_active`	Can be used to retrieve active or inactive results only e.g `GET /api/facilities/facilities/?is_active=false`

Note: Filters can be combined / chained.

Documentation examples

All the examples in this documentation will use the recommended JSON format.

Data notations

The example below demonstrates the manner in which example JSON payloads in the documentation should be interpreted:

```
{
    "name": "John Doe",
    "gender": "M",
    "age": 33,
    "houses": [
        {
            "city": "Nairobi",
            "type": "Flat"
        },
        {
            "city": "Mombasa",
            "type": "Bungalow"
        }
    ],
    "phone": {
        "work": "8781923",
        "home": "213789123"
    }
}
```

This table describes the data above

Property	Type	Description
name	string	Name of the person
age	integer	Age of the person
gender	string	Gender of the person
houses	list of objects	A list of houses the person owns
houses[ ].city	string	The city in which the house is in
houses[ ].type	string	The type of the house
phone	object	The person's phone numbers
phone.work	string	Work phone number
phone.home	string	Home phone number

The *[ ]* notation is used to indicate a property of every object in a list. For example, `houses[ ].city` means every object in the list `houses` has a property called `city`.

Data types

The data types are standard JSON. The MFL API uses UUIDs for its primary keys.

Data type	JSON Representation	Description
string	string	A sequence of zero or more characters wrapped in double quotes.
object	object	A collection of name-value pairs wrapped in curly braces : { and }
list	array	A collection of values
boolean	boolean	Represents truthy values and falsy values.Valid values are `true` and `false`
null	null	Represents null values
integer	integer	Integer values
decimal	string	Precision decimal values represented as strings
uuid	string	A string of 32 characters used as a unique identifier (UUIDs)
datetime	string	A string representing date and time values (*Dates and times*)
url	string	A string representing the location of a network resource

URLs

URLs in this document shall be written in shortform, excluding the scheme and domain (or IP) from which MFL can be accessed.

For a production system, the scheme shall always be `https`, unless otherwise specified.

For example, if MFL is running from the IP *192.168.1.56*, a full URL could be `https://192.168.1.56/api/common/contacts/`. In the documentation, the URL shall be written as `/api/common/contacts/`, exluding the scheme and domain (or IP).

Note: **All URLs have a trailing slash** unless specified otherwise. For example, the url `https://192.168.1.56/v1/claims/` is not equivalent to the url `https://192.168.1.56/v1/claims`. The latter will result in a `HTTP 404` (Not Found) response

---

URL Parameters

Any API endpoints that support url parameters shall be specified in the following format:

```
/api/common/counties/<value>/
```

For example to retrieve a county by its ID (UUID), the URL shall be specified as:

```
/api/common/counties/<id>/
```

e.g. `/api/common/counties/89d8f3dd698b46e6a052f355a231858d/`

URL Query Parameters

Any API endpoints that support query parameters shall be specified in the following format:

```
/api/common/counties/?name=<value>
```

For example to query the county endpoint by `name`, the URL shall be specified as:

```
/api/common/counties/?name=<name>
```

e.g. `/api/common/counties/?name=Nairobi`

Dates and times

All dates and times shall be represented as datetime strings in ISO 8601 format i.e.

```
YYYY-MM-DDTHH:MM:SSZ
```

e.g. `2015-03-30T15:23:89Z`

If timezone is to be included, the timezone shall be `UTC`, thus the format becomes

```
YYYY-MM-DDTHH:MM:SS+0000
```

e.g. `2015-03-30T15:23:89+0000`

Any date that does not have a timezone shall be assumed to be `UTC`.

UUIDs

UUIDs are used as unique record identifiers for each record in MFL. All UUIDs used in MFL are version 4 UUIDs.

HTTP Errors

400 (Bad Request) This error occurs if the request given to the server is malformed or does not meet certain criteria e.g. invalid data.

401 (Unauthorized) The request to access a resource was unauthorized. (*Authentication and authorization*)

403 (Forbidden) The authorized user does not have permission to access a resource (*Authentication and authorization*)

404 (Not found) The requested resource was not found

410 (Gone) The requested resource has been removed

500 (Server Error) A server error has occurred

Pagination

Endpoints that return multiple items will be paginated with a page size of 25 by default. All endpoints returning a list of items shall have the following format:

```
GET /api/common/constituencies/?page=2
```

```
{
"count": 290,
"next": "http://localhost:8000/api/common/constituencies/?page=3",
"previous": "http://localhost:8000/api/common/constituencies/",
"results": [
    {
        ... // list of items requested, in this case constituencies
    ]
}
```

A client can request a larger page size by specifying the `page_size` parameter e.g `/api/common/contacts/?page_size=100`. There page size limit is selected at server configuration time; it will usually be around 1000 items.

Audit trail

The API server provides an audit trail for all non third-party resources. This audit trail can be accessed on **detail endpoints** by appending an `include_audit=true` query parameter.

For example, if there was a contact with the id `28d2a0c8-40f4-4686-97d0-d7c6f453fcb3`, a `GET` request to `/api/common/contacts/28d2a0c8-40f4-4686-97d0-d7c6f453fcb3/?include_audit=true` would return a payload that has a *revisions* key that contains a representation of every past revision of that specific contact.

Search

Every **list** endpoint supports **full text** search. Search is implemented as a filter, using the *search* query parameter.

For example, to search for contacts that have the word "meru" in them, the query would be `/api/common/contacts/?search=meru`.

The API sandbox

Our experience teaches us that the biggest roadblock to systems integration is usually communication. Developers operate at a level of precision and detail that is alien to most people. We've been spoilt by our past dabblings with high quality API documentation sites like the Stripe API site.

Swagger

The API can be interacted with through Swagger from the link `api/explore/`

The Browsable API

The API is accesible from the URL `api/`. This is the entry point into the entire list of all the URLs in the API and the methods and that are allowed on an endpoint.

API Metadata support

The `api/` URL has been designed to make it easy for a client accessing an endpoint to know the methods that are allowed on the endpoint. The metadata support also allows a client to know the fields that an endpoint accepts and whether they are required or not.

Metadata resources

This chapter assumes that the reader is familiar with the general principles explained in the *Using the API - basic principles* chapter.

The MFL's job description is to standardize the management of information relating to facilities (including community health units), provide a standard catalogue of available healthcare *The service catalog* and act as a central ingress point for regulation. However, in order to do this, the MFL needs to have a constellation of support resources in its data model.

This chapter concerns itself with the resources that hold "setup" type information. These resources wil often be used to populate drop-downs and other types of choosers in the web / mobile front-ends.

Contact Types

The contact type resource allows us to move the configuration of contact types that are recognized by the server from code to configuration.

This API will typically be used by web front-ends that need to populate contact type selection dropdowns during the creation of contacts/

The contact type list endpoint is at `/api/common/contact_types/` while the detail endpoint will be at `/api/common/contact_types/<pk>/` (for example, the contact whose id is `3a05b4e7-fb8e-4c23-ac95-4e36ac2b99fa` can be retrieved by `GET``ting ``/api/common/contact_types/3a05b4e7-fb8e-4c23-ac95-4e36ac2b99fa/`).

When creating a new contact, the only necessary fields are the `name` and `description`. The following is a valid `POST` payload:

```
{
    "name": "KONTACT TYPE",
    "description": "Documentation Example"
}
```

Towns

The town resource allows us to set up the system's list of towns.

This API will typically be used by front-ends that need to populate town selection dropdowns during the creation of facility records.

The town list endpoint is at `/api/common/towns/`. As with every other resource in this API, the detail endpoint will be at `/api/common/towns/` e.g `/api/common/towns/e8f369f1-d115-43a1-a19b-ae40b7b4b19e/` for a town whose primary key is `e8f369f1-d115-43a1-a19b-ae40b7b4b19e`.

When creating a new town, the only mandatory parameter is the `name`. The following is a valid `POST` payload:

```
{
    "name": "Documentation Town"
}
```

Administrative units

The second generation MFL implements the post 2010 (Kenyan) constitution administrative structure. This structure has only three levels, after the national one: counties, constituencies and wards.

There are 47 counties. Each county contains a number of constituencies - all adding up to 290. Each constituency in turn contains a number of wards - all adding up to 1450.

The constituencies will sometimes be referred to as "sub-counties". The wards often - ut not always - correspond to locations in the previous administrative structure.

It is unlikely that an API client will need to alter the administrative unit setup data (it is part of the server's default data). API support for editing has still been supplied - as a failsafe mechanism.

Counties

Counties can be listed by visiting `/api/common/counties/`. Individual county details can be listed by visiting `/api/common/counties/<pk>/` e.g `/api/common/counties/dd999449-d36b-47f2-a958-1f5bb52951d4/` for a county whose `id` is `dd999449-d36b-47f2-a958-1f5bb52951d4`.

Note: The county detail endpoint is atypical.

It embeds a geographic feature (GeoJSON) under the `county_boundary` key and the coordinates of all facilities (as a map of GeoJSON points) in the county under the key `facility_coordinates`.

This API provides all the raw information that is needed to render a map of the county and plot the facilities on that map.

Constituencies

Constituencies can be listed by visiting `/api/common/constituencies/`. Individual constituency details can be viewed by visiting `/api/common/constituencies/<pk>/` e.g `/api/common/constituencies/16da4d8a-4bff-448b-8fbb-0f64ee82c05a/` for the constituency with an `id` `16da4d8a-4bff-448b-8fbb-0f64ee82c05a`.

Note: Like the county detail endpoint, the constituency detail endpoint is atypical. It embeds the same coordinates and boundary information.

Wards

Wards can be listed by visiting `/api/common/wards/`. Individual ward details can be retrieved at `/api/common/wards/<pk>/` e.g `/api/common/wards/41ae635c-5dba-40af-bb74-37d8d0a4c175/` for the ward with an `id` `41ae635c-5dba-40af-bb74-37d8d0a4c175`.

Note: Like the county and constituency detail endpoints, the ward detail endpoint is atypical because it embeds coordinates and boundary information.

Facility Types

The purpose of this resource is to populate dropdowns used in facility creation and edit screens. The API also supports the creation of an administrative interface that can be used to add new facility types and retire old ones.

Facility types can be listed at `/api/facilities/facility_types/`. Individual facility details can be listed at `/api/facilities/facility_types/<pk>/` e.g `/api/facilities/facility_types/ccf14e50-2606-40b9-96fd-0dc5b3ed4a15/` for the facility whose `id` is `ccf14e50-2606-40b9-96fd-0dc5b3ed4a15`.

The only required fields when creating a new facility type are `name` (which should be set to something meaningful) and `sub_division` (which can be null). The following is a minimal but valid `POST` payload:

```
{
    "name": "Test facility type for docs",
    "sub_division": null
}
```

Facility owners and owner types

Facility owner types provide a mechanism by which the owners of facilities can be classified, arbitrarily. Examples are "Non Governmental Organizations", "Faith Based Organizations" and the "Ministry of Health". These owner types can be changed at will.

In the MFL 1 era, facility owners were set up in a very general manner e.g "Private Enterprise (Institution)" and "Private Practice - Unspecified". There is no technical reason why these facility owners cannot be more specific e.g names of specific private sector organizations.

Facility owner types

Facility owner types can be listed at `/api/facilities/owner_types/`. Predictably, the detailed representations will be found at `7ce5a7b1-9a5e-476c-a01c-8f52c4233926`.

When creating a new facility owner type, the only mandatory field is the `name`. For example: the following is a perfectly valid `POST` payload:

```
{
    "name": "Owner type for docs"
}
```

Facility owners

Facility owners can be listed at `/api/facilities/owners/`. Detail representations can be obtained from `/api/facilities/owners/<pk>/` e.g `/api/facilities/owners/f770a132-f62a-418a-96b4-062c3cc7860c/`.

When registering a new facility owner, the `POST` payload should contain at least the `name`, `description` and `abbreviation`. For example:

```
{
    "name": "Imaginary BigCorp.",
    "description": "BigCorp owns everything",
    "abbreviation": "BIG",
}
```

Note: The setup of owners and owner types should be performed with care, because of the importance of this metadata in analysis / reporting.

Job titles

The job titles that are available to be assigned to facility officers can be listed at `/api/facilities/job_titles/`. Individual job title detail resources will be at `/api/facilities/job_titles/<pk>` e.g `/api/facilities/job_titles/7ec51365-75b7-45e5-873b-8bb3c97bbe21`.

When creating a new job title, the `name` and `description` should be sent as a `POST` payload to the list endpoint. The example below is a valid payload:

```
{
    "name": "Boss",
    "description": "Big Cahunna"
}
```

Regulating bodies

The regulators that are known to the server can be listed by `GET``ting ``/api/facilities/regulating_bodies/`. Predictaby, the detail of each can be retrieved at `/api/facilities/regulating_bodies/<pk>/` e.g `/api/facilities/regulating_bodies/07f8302f-042a-4a9c-906b-10d69092b43e/`.

When registering a new regulating body, you should set the `name`, `abbreviation` and `regulation_verb` fields. For example:

```
{
    "name": "A newly legislated regulator",
    "abbreviation": "ANLR",
    "regulation_verb": "Gazettment"
}
```

Regulating body contacts

After creating a regulating body, one or more contacts can be associated with it by `POST``ing to ``/api/facilities/regulating_body_contacts/` the `id` of the `regulating_body` ( returned by the API after creating the body or retrieved from the relevant list / detail endpoint ) and the `id` of the `contact` (obtained in a similar manner).

Suppose that the `id` for the newly created regulating body is `5763a053-668e-4ca7-bab4-cda3da396453`. Suppose also that we have just created a contact with `id` `7dd62ab9-94c2-48d6-a10f-d903bd57acd5`.

We can associate that contact and the regulating body by `POST``ing to ``/api/facilities/regulating_body_contacts/` the following payload:

```
{
    "regulating_body": "5763a053-668e-4ca7-bab4-cda3da396453",
    "contact": "7dd62ab9-94c2-48d6-a10f-d903bd57acd5"
}
```

The regulating body contacts that already exist can be listed by issuing a `GET` to `/api/facilities/regulating_body_contacts/`. If you would like to filter those that belong to a known regulating body, use a `regulating_body` query parameter, with the `id` of the regulating body as the filter value e.g `/api/facilities/regulating_body_contacts/?regulating_body=5763a053-668e-4ca7-bab4-cda3da396453`. You could also filter the regulating body contacts using the `id` of a known contact, although the use cases for that are more limited.

Note: This section introduces some patterns that will recur in this API:

- The use of filters: the list APIs are filterable by most of the fields that they list. You can explore this further in the *The API sandbox*.
- The use of **explicit join tables** for many to many relationships.

The `regulating_body_contact` resource that is the subject of this section is an example. That is a deliberate choice - we find that, even though it makes the API clients do a little more work, it leads to more **reliable** APIs. In RESTful APIs, nested serialization / deserialization is a massive pain. We'd rather not deal with it.

Facility Operation Status

Operation Status is what indicates whether a facility is operation or non operational.

Listing Available Operation Status

To list all the available operation status in MFL do a `GET` to the URL `api/facilities/facility_status/`

Sample Expected Response data:


```
{
    "count": 4,
    "next": null,
    "previous": null,
    "page_size": 30,
    "current_page": 1,
    "total_pages": 1,
    "start_index": 1,
    "end_index": 4,
    "results": [
        {
            "id": "7e5cfa76-7564-4263-89d6-c4e30ce64b39",
            "created": "2015-09-28T08:59:03.979532Z",
            "updated": "2015-09-28T08:59:03.979567Z",
            "deleted": false,
            "active": true,
            "search": null,
            "name": "Unknown",
            "description": null,
            "created_by": 1,
            "updated_by": 1
        },
        {
            "id": "c3b2f2f3-2cfa-4203-bc92-476f63069377",
            "created": "2015-09-28T08:59:03.973257Z",
            "updated": "2015-09-28T08:59:03.973276Z",
            "deleted": false,
            "active": true,
            "search": null,
            "name": "Pending Opening",
            "description": null,
            "created_by": 1,
            "updated_by": 1
        },
        {
            "id": "c879932e-4763-420a-9a87-adddb874b662",
            "created": "2015-09-28T08:59:03.967246Z",
            "updated": "2015-09-28T08:59:03.967266Z",
            "deleted": false,
            "active": true,
            "search": null,
```

```
            "name": "Not-Operational",
            "description": null,
            "created_by": 1,
            "updated_by": 1
        },
        {
            "id": "d498f6bb-af28-435d-b83c-39e81421a83c",
            "created": "2015-09-28T08:59:03.957568Z",
            "updated": "2015-09-28T08:59:03.957590Z",
            "deleted": false,
            "active": true,
            "search": null,
            "name": "Operational",
            "description": null,
            "created_by": 1,
            "updated_by": 1
        }
    ]
}
```


Expected Response Code: `HTPP_200_OK`

Creating an Operation Status

To add a new operation status `POST` to the URL `api/facilities/facility_status/` a payload similar to the one shown below:


```
{
    "name": "Test Status",
    "description": "This is just for testing"
}
```


Sample Expected Response data:


```
{
    "id": "4a67f0f4-bc3a-461f-ad26-4aad885482f4",
    "created": "2015-10-27T08:19:55.764752Z",
    "updated": "2015-10-27T08:19:55.764767Z",
    "deleted": false,
    "active": true,
    "search": null,
    "name": "Test Status",
    "description": "This is just for testing",
    "created_by": 4,
    "updated_by": 4
}
```


Expected Response Code: `HTTP_201_CREATED`

Updating a single Operation Status

To update a single operation status do a `PATCH` to the URL `api/facilities/facility_status/<facility_status_id>/`

For example to update operation status we just created we would do a PATCH to the URL `api/facilities/facility_status/4a67f0f4-bc3a-461f-ad26-4aad885482f4/` with a payload similar to the one below:

```
{
    "name": "Test Status edited"
}
```

Sample Expected Response data

```
{
    "id": "4a67f0f4-bc3a-461f-ad26-4aad885482f4",
    "created": "2015-10-27T08:19:55.764752Z",
    "updated": "2015-10-27T08:19:55.764767Z",
    "deleted": false,
    "active": true,
    "search": null,
    "name": "Test Status edited",
    "description": "This is just for testing",
    "created_by": 4,
    "updated_by": 4
}
```

Expected Response code: `HTTP_200_OK`

The service catalog

This chapter assumes that the reader is familiar with the general principles explained in the *Using the API - basic principles* chapter.

In order for the MFL to do its job as the keystone of the Kenyan national health information system, there needs to be a standard registry of services.

At the time when this edition of the MFL was built, no such thing existed. The MFL therefore took on the responsibility of providing that registry.

This chapter concerns itself with the setup of the **service catalog**. The service catalog has two primary goals:

- to model healthcare services in a manner that is flexible and future proof
- to standardize service codes

Note: Standardization of service codes is a pre-requisite for interoperability between the MFL and other systems.

Note: The flexibility will allow the MFL to keep pace with changes in healthcare and policy.

Service Categories

Service categories are the "broad headings" under which healthcare services are classified. An example is "Comprehensive Emergency Obstetric Care (CEOC)", an umbrella for services that respond to life-threatening emergency complications and are offered by facilities whose human resources include doctors and whose infrastructure includes operating theatres and incubators.

Existing service categories can be listed by issuing a `GET` to `/api/facilities/service_categories/`.

To add a new service category, `POST` to the same URL a payload similar to this:

```
{
    "name": "A new service category",
    "description": "What is it really about",
    "abbreviation": "ABBR"
}
```

Services

Services are the granular "product" delivered to end users. Some examples are "Provider Initiated Counselling and Testing" and Oral Health Services (Dental Services)".

Existing services can be listed at `/api/facilities/services/`.

When creating a new service, `POST` the `name`, `description`, `abbreviation` and `category`. For example:

```
{
    "name": "A new service",
    "description": "The best new service since bread slicing",
    "abbreviation": "ANS",
    "category": "2bdfd814-5cba-4673-916e-96b6a98cf1c9"
}
```

Note: Services get auto-assigned `code` s. A service code is immutable once issued. The service codes are expected to become a standard identifier for services.

Options and service options

In order to understand the options API, we'll take a look at the Facility Creation Form from the 2010 Master Facility List Implementation Guide (the guiding document for the previous edition of the MFL).

In the form above, many services have `Yes` and `No` options. Some services require a numeric level (levels `1` to `6` from the Kenya Essential Package for Health [KEPH]), while obstetric services are classified into `Basic` or `Comp` (comprehensive).

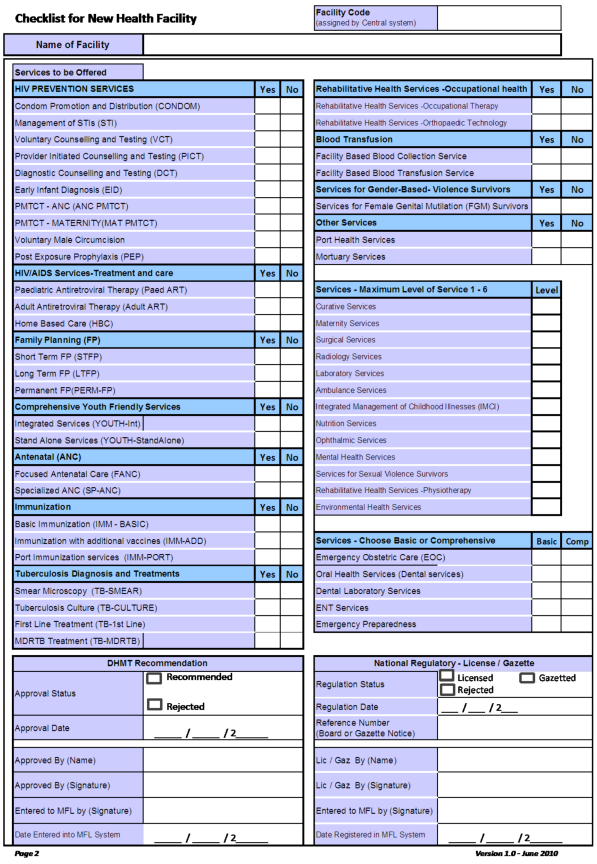
That form is far from comprehensive (that was found out in practice). A naive implementation of that form would hobble the system if a new standard service catalog emerged.

This API responds to that challenge by creating a mechanism by which a service can be associated with an arbitrary range of options.

Note: This approach will make API clients (including the official web front-ends) do a lot more work; but in this case, we think that it is worthwhile.

Options

"Options" are the possible "choices" in a service questionnare, like the one shown above.



Checklist for New Health Facility

Facility Code (assigned by Central system)	
Name of Facility	

Services to be Offered

HIV PREVENTION SERVICES	Yes	No
Condom Promotion and Distribution (CONDOM)		
Management of STIs (STI)		
Voluntary Counselling and Testing (VCT)		
Provider Initiated Counselling and Testing (PICT)		
Diagnostic Counselling and Testing (DCT)		
Early Infant Diagnosis (EID)		
PMTCT - ANC (ANC PMTCT)		
PMTCT - MATERNITY(MAT PMTCT)		
Voluntary Male Circumcision		
Post Exposure Prophylaxis (PEP)		
HIV/AIDS Services-Treatment and care	**Yes**	**No**
Paediatric Antiretroviral Therapy (Paed ART)		
Adult Antiretroviral Therapy (Adult ART)		
Home Based Care (HBC)		
Family Planning (FP)	**Yes**	**No**
Short Term FP (STFP)		
Long Term FP (LTFP)		
Permanent FP(PERM-FP)		
Comprehensive Youth Friendly Services	**Yes**	**No**
Integrated Services (YOUTH-Int)		
Stand Alone Services (YOUTH-StandAlone)		
Antenatal (ANC)	**Yes**	**No**
Focused Antenatal Care (FANC)		
Specialized ANC (SP-ANC)		
Immunization	**Yes**	**No**
Basic Immunization (IMM - BASIC)		
Immunization with additional vaccines (IMM-ADD)		
Port Immunization services (IMM-PORT)		
Tuberculosis Diagnosis and Treatments	**Yes**	**No**
Smear Microscopy (TB-SMEAR)		
Tuberculosis Culture (TB-CULTURE)		
First Line Treatment (TB-1st Line)		
MDRTB Treatment (TB-MDRTB)		

Rehabilitative Health Services -Occupational health	Yes	No
Rehabilitative Health Services -Occupational Therapy		
Rehabilitative Health Services -Orthopaedic Technology		
Blood Transfusion	**Yes**	**No**
Facility Based Blood Collection Service		
Facility Based Blood Transfusion Service		
Services for Gender-Based- Violence Survivors	**Yes**	**No**
Services for Female Genital Mutilation (FGM) Survivors		
Other Services	**Yes**	**No**
Port Health Services		
Mortuary Services		

Services - Maximum Level of Service 1 - 6	Level
Curative Services	
Maternity Services	
Surgical Services	
Radiology Services	
Laboratory Services	
Ambulance Services	
Integrated Management of Childhood Illnesses (IMCI)	
Nutrition Services	
Ophthalmic Services	
Mental Health Services	
Services for Sexual Violence Survivors	
Rehabilitative Health Services -Physiotherapy	
Environmental Health Services	

Services - Choose Basic or Comprehensive	Basic	Comp
Emergency Obstetric Care (EOC)		
Oral Health Services (Dental services)		
Dental Laboratory Services		
ENT Services		
Emergency Preparedness		

DHMT Recommendation	
Approval Status	☐ Recommended ☐ Rejected
Approval Date	_____ / _____ / 2_____
Approved By (Name)	
Approved By (Signature)	
Entered to MFL by (Signature)	
Date Entered into MFL System	_____ / _____ / 2_____

National Regulatory - License / Gazette	
Regulation Status	☐ Licensed ☐ Gazetted ☐ Rejected
Regulation Date	___ / ___ / 2___
Reference Number (Board or Gazette Notice)	
Lic / Gaz By (Name)	
Lic / Gaz By (Signature)	
Entered to MFL by (Signature)	
Date Registered in MFL System	_____ / _____ / 2_____

Page 2 *Version 1.0 - June 2010*

79

Using that example: "Yes" and "No" are options for the services under the "HIV Prevention Services" category, while the numbers "1,2,3,4,5,6" are options for the KEPH service classification section.

The known service options can be listed and created at `/api/facilities/options/`. To create a new option, you need to `POST` a payload that includes the following fields:

Field	Description
value	The value that will be stored in the database, and analyzed. This should be a constant that is friendly to analytical tools e.g one that does not have unnecessary punctuation and spacing. This will be a string.
display_text	The description that will be displayed to the user wherever the option appears in the user interface. This should be plain text. It cannot be blank.
is_exclusive_option	This is a **boolean** value; if `true`, only one of the exclusive options can be selected for a specific facility and service. A user interface should intepret this by implementing a control that behaves like radio buttons.
option_type	The choices are `BOOLEAN`, `INTEGER`, `DECIMAL` and `TEXT`. This controls the type of response data that is valid for that option.

Here is an example of a valid `POST` payload:

```
{
    "value": "YES",
    "display_text": "Yes",
    "is_exclusive_option": true,
    "option_type": "BOOLEAN"
}
```

Service Options

The service options resource is used to link services and options. To use an example from the form above, the service "Home Based Care (HBC)" should be linked with the options `Yes` and `No`. Service options can be viewed and configured at `/api/facilities/service_options/`. To create a new link, you need to know the `id` of the service and the option.

For example: to link an option with the `id` `53c3f729-97d1-4c9d-9fff-d2edc797b185` with the service with the `id` `80613650-f765-4032-a9d3-bb0fc9cc37cc`, `POST` to `/api/facilities/options/` the following payload:

```
{
    "service": "80613650-f765-4032-a9d3-bb0fc9cc37cc",
    "option": "53c3f729-97d1-4c9d-9fff-d2edc797b185"
}
```

Linking facilities to services

The final step is to link a facility to the services that it offers. **Facilities are linked to services through service options.**

If the service option that we created above has the `id` `f09af53e-5c6f-468d-a41d-df51693e51a3` and we'd like to link it to a facility whose `id` is `c4169b23-5cbb-4ed8-a556-8a4fc43af17e`, `POST` to `/facilities/facility_services/` the following payload:

```
{
    "facility": "c4169b23-5cbb-4ed8-a556-8a4fc43af17e",
```

```
    "selected_option": "f09af53e-5c6f-468d-a41d-df51693e51a3"
}
```

Facilities

This chapter assumes that the reader is familiar with the general principles explained in the *Using the API - basic principles* chapter.

The MFL is not merely a "list" of facilities; it has rich APIs to manage their life cycles and to support interaction with other healthcare systems. This chapter concerns itself with what is arguably the "core" of the MFL system - the maintenance of facility information. Facilities APIs fall into the following groups:

Function	Resources / APIs
Facility Information storage	• Facility • Facility Physical Addresses • Facility Contacts • Facility Units • Facility Services
Facility Workflow / Life cycle	• Approval • Publishing (synchronization) • Regulation • Upgrade • Downgrade
Facility ratings	• Facility service ratings • Facility ratings report
Facility downloads	• Facility cover letters • Facility correction templates • Facility excel exports
Facility dashboard APIS	• Analysis by administrative units • Analysis by type • Analysis by owner and owner category • Analysis by regulator and regulation status

Note: One of the things associated with facilities that are registered on the Master Facilities List is a **Master Facilities List (MFL) Code**.

The MFL code is a unique number (integral) that is *sequential* and *immutable*. The immutability is taken seriously - the MFL codes that were issued under the first generation system will not be re-issued under the second generation MFL system.

Codes that are issued under MFL 2 will start at *100000*.

Facility information storage

Note: These APIs are the "heart" of the MFL system. A well-behaved front-end should take an integrated approach, presenting output from these APIs under one set of screens (instead of five sets, one for each resource type).

Facilities

Listing multiple records

The facilities that are currently registered can be listed at `/api/facilities/facilities/`.

Retrieving a single record

Each facility has a UUID `id`. A facility's detail record can be listed at `/api/facilities/facilities/<id>/`. For example: if a facility record's `id` is `2927d31f-b1a0-4d17-93b0-ea648af7b9f0`, the detail URL for the facility record will be `/api/facilities/facilities/2927d31f-b1a0-4d17-93b0-ea648af7b9f0/`.

Filtering and search

Facilities - as listed at `/api/facilities/facilities/` can be filtered using the following:

Filter	Explanation
name	This does a *case insensitive partial match* but accepts only one name to filter by e.g `/api/facilities/facilities/?name=molo`.
code	Filter by **one or more** facility codes e.g `/api/facilities/facilities/?code=15003,15002`. The `,` is used to separate individual parameters. This does exact matches.
description	Similar to `name` but operating on descriptions e.g `/api/facilities/facilities/?name=molo`
facility_type	Filter by the `id``s of one or more facility types e.g ``/api/facilities/facilities/?facility_type=f25ba517-3b8d-4692-ba7b-3524f6ec58e5,b2225473-08f1-4e86-a47a-0a61cf75e731`. Facility types can be listed at `/api/facilities/facility_types/`.
operation_status	Filter by the `id` of one or more operation statuses from `/api/facilities/facility_status/`
ward	Filter by the `id` of wards (from `/api/common/wards/` e.g `/api/facilities/facilities/?ward=353404d7-02e6-422f-b64f-b1c7d0f1bcf0`)
county	Filter by the `id` of counties (from `/api/common/counties/` e.g `/api/facilities/facilities/?county=fa47afa2-a78a-421f-ad9f-55e6cbfc280c`)
constituency	Filter by the `id` of constituencies (from `/api/common/constituencies/` e.g `/api/facilities/facilities/?constituencies=93280ce0-670f-4b96-a449-57d65f0dcace`)
owner	Filter by the `id` of one or more owners. Owners can be listed at `/api/facilities/owners/`
owner_type	Filter by the `id` of one or more owner types. Owner types can be listed at `/api/facilities/owner_types/`
officer_in_charge	Filter by the `id` of one or more officers-in-charge. The officers can be listed at `/api/facilities/officers/`
number_of_beds	Filter by the number of beds, supplying one or more filter parameters e.g `/api/facilities/facilities/?number_of_beds=20,21,22,23,24`
number_of_cots	Filter by the number of cots, supplying one ormore filter parameters e.g `/api/facilities/facilities/?number_of_cots=10,11,12`
open_whole_day	A boolean filter e.g `/api/facilities/facilities/?open_whole_day=true`
open_whole_week	A boolean filter e.g `/api/facilities/facilities/?open_whole_week=true`
is_classified	A boolean filter that determines if a facility's coordinates should be shown or not. The public front-end should omit classified facilities by default. i.e. publish those that can be listed with `/api/facilities/facilities/?is_classified=false`
is_published	A boolean filter that determines if a facility has been cleared for display on the public site. The public site should only display facilities that can be listed with `/api/facilities/facilities/?is_published=true`
is_regulated	The facilities that are pending action from the regulators can be listed with `/api/facilities/facilities/?is_regulated=False`

The following filters are common to **all** list endpoints in this API, other than `/api/users/`.

Filter	Explanation
updated_before	The most recently updated facilities can be listed with a query similar to `/api/facilities/facilities/?updated_before=2015-05-09T08:57:48.094112Z`. The datetime is in ISO 8601 format.
created_before	Similar to `updated_before`, but operating on creation dates. Creation dates are not "touched" after the initial creation of the resource.
updated_after	Similar to `updated_before`, but returns records newer than the specified datetime
created_after	Similar to `updated_after`, but works with creation dates.
updated_on	This is similar to the date filters above but performs **exact matches** on the update date.
created_on	This is also performs exact matches.
is_active	For all resources in this server, the preferred way to "retire" records is to mark them as inactive. This allows the API client to request only active or only inactive records.
search	Perform a full text search that looks through all fields. e.g `/api/facilities/facilities/?search=endebess` gives back all facilities that have "endebess" anywhere in their name, description or attributes.

Note: These filters can be combined / chained.

For example: `/api/facilities/facilities/?ward=353404d7-02e6-422f-b64f-b1c7d0f1bcf0&open_whole_`

Adding a new record

The following are the important fields when adding a new facility:

Field	Explanation
name	The name of the faciity e.g "Musembe Dispensary (Lugari)"
abbreviation	A shortened name
description	Free text that supplies any additional detail that is required
location_desc	An explanation of the location, in "plain" language e.g "Eldoret - Webuye Highway (at Musembe Mkt junction)"
number_of_beds	The number of beds as per the facility's license
number_of_cots	The number of cots as per the facility's license
open_whole_day	*true* if the facility is a 24 hour operation
open_whole_week	*true* if the facility is a 7 day operation
facility_type	An `id`, obtained by listing `/api/facilities/facility_types/`
operation_status	An `id`, obtained from `/api/facilities/facility_status/`. This is the overall state of the facility e.g "Operational" or "Not Operational"
ward	An `id`, obtained from `/api/common/wards/`. Facilities are attached at the level of the smallest administrative area (the ward).
owner	An `id`, obtained from `/api/facilities/owners/`.
officer_in_charge	An `id`, obtained from `/api/facilities/officers/`
physical_address	An `id`, obtained from `/api/common/address/`
parent	Optional. If a facility is a "branch" of a larger facility, the `id` of the parent facility should be supplied here.

The following example illustrates a valid `POST` payload:

```
{
    "name": "Demo Facility",
    "abbreviation": "DEMOFAC",
    "description": "This is an example in the documentation",
    "location_desc": "Planet: Mars",
    "number_of_beds": 20,
    "number_of_cots": 0,
    "open_whole_day": true,
    "open_whole_week": true,
    "facility_type": "db8f93ad-b558-405a-89b5-a0cdb318ee6e",
    "operation_status": "ee194a52-db9d-401c-a2ef-9c8225e501cd",
    "ward": "a64d930d-883e-4b96-ba39-c792a1cd04f2",
    "owner": "f4c7ca47-7ee6-4795-ac1c-a5d219e329ad",
    "officer_in_charge": "972c9c96-fe27-4803-b6f8-c933310e2f44",
    "physical_address": "88dde94b-dc42-4b13-b1cb-05eca047678c",
    "parent": null
}
```

A successful `POST` will get back a `HTTP 201 Created` response. A representation of the freshly created resource will be returned in the response.

Updating an existing record

In order to update an existing record, `PATCH` the appropriate field from the record's detail view.

For example, if the facility that we created above got the `id` set to `e88f0c1a-e1e4-44ff-8db1-8c4135abb080` (this will be returned to the client in the resource re-

turned after successful creation), we can change its `location_desc` from "Planet: Mars" to "Planet: Venus" by sending a `PATCH` to `/api/facilities/facilities/e88f0c1a-e1e4-44ff-8db1-8c4135abb080/` with the following payload:

```
{
    "location_desc": "Planet: Venus"
}
```

A successful `PATCH` will get back a `HTTP 200 OK` response and a representation of the freshly updated resource will be returned.

Deleting a record

In order to delete the record that we just created, send a `DELETE` with an empty payload to the detail URL i.e. to `/api/facilities/facilities/e88f0c1a-e1e4-44ff-8db1-8c4135abb080/` in the example above.

A successful `DELETE` will get back a `HTTP 204 NO CONTENT` response.

Physical addresses

Listing multiple records

The physical addresses known to the system can be listed at `/api/common/address/`.

In addition to the common filters that are already explained above, physical addresses have the following extra filters:

Field	Explanation
town	Filter by the `id` of a town. Towns can be listed at `/api/common/towns/` e.g `/api/common/address/?town=b2af0361-c924-4ba2-9bc6-82333fc0a26f`
postal_code	Filter by the `postal_code` e.g `/api/common/address/?postal_code=00100`
address	Filter by the actual text of the address itself e.g `/api/common/address/?address=P.O.%20Box%201`
nearest_landmark	Filter by the contents of the `nearest_landmark` field e.g `/api/common/address/?nearest_landmark=kicc`
plot_number	Filter by the `plot_number` field e.g `/api/common/address/?plot_number=940`

Retrieving a single record

The detail endpoint is `/api/common/address/<id>/` e.g `/api/common/address/20d01a89-f6b5-4a4d-b788-32182d427c18/` for the address whose `id` is `20d01a89-f6b5-4a4d-b788-32182d427c18`.

Adding a new record

Supply the following fields:

Field	Explanation
postal_code	A valid postal code e.g "00100"
address	An address e.g "No. 11A, Kabarnet Court, off Kabarnet Road" or "P.O. Box 5980"
nearest_landmark	Free text, left to the discretion of the person creating the record
plot_number	Free text, left to the discretion of the person entering the record
town	The `id` of a town, as listed at `/api/common/towns/`

```
{
    "postal_code": "00100",
    "address": "No. 11A, Kabarnet Court, off Kabarnet Road",
    "nearest_landmark": "Kingdom Business Centre",
    "plot_number": "-",
    "town": "b2af0361-c924-4ba2-9bc6-82333fc0a26f"
}
```

A successful `POST` will get back a `HTTP 201 Created` response. A representation of the freshly created resource will be returned in the response.

Updating an existing record

`PATCH` the detail endpoint above e.g to set the `plot_number` for the example record above, send the following `PATCH` payload to `/api/common/address/20d01a89-f6b5-4a4d-b788-32182d427c18/`:

```
{
    "plot_number": "250"
}
```

A successful `PATCH` will get back a `HTTP 200 OK` response. A representation of the updated resource will be returned in the response.

Deleting a record

Send a `DELETE` request to the detail endpoint. In the example above, the `DELETE` would be sent to `/api/common/address/20d01a89-f6b5-4a4d-b788-32182d427c18/`.

A successful `DELETE` will get back a `HTTP 204 NO CONTENT` response.

Facility contacts

Listing multiple records

Facility contacts can be listed at `/api/facilities/contacts/`.

In addition to the common contacts that are already explained above, facility contacts have the following extra fields:

Field	Explanation
facility	The `id` of the relevant facility, as listed at `/api/facilities/facilities/` e.g `/api/facilities/contacts/?facility=faaefb75-dba4-4564-8acb-6b947685de24`
contact	The `id` of a contact, as listed at `/api/common/contacts/` e.g `/api/facilities/contacts/?contact=2f5fe4c2-0371-4ba0-ba31-79d997d71c6a`

Retrieving a single record

The detail endpoint is `/api/facilities/contacts/<id>/`. For example, the detail URL for the facility contact whose `id` is `9641f588-a5c0-4c0d-ad13-cfcf98a2fb7` is `/api/facilities/contacts/9641f588-a5c0-4c0d-ad13-cfcf98a2fb7`.

Adding a new record

The only required fields are the `facility` and `contact` (as documented above).

The following example is a valid `POST` payload:

```
{
    "facility": "faaefb75-dba4-4564-8acb-6b947685de24",
    "contact": "2f5fe4c2-0371-4ba0-ba31-79d997d71c6a"
}
```

A successful `POST` will get back a `HTTP 201 Created` response. A representation of the freshly created resource will be returned in the response.

Updating an existing record

`PATCH` the detail endpoint with the new values e.g to change the contact in the record above, a valid `PATCH` payload could be:

```
{
    "contact": "516f64b5-a12c-4323-b918-a5512b4baf6a"
}
```

A successful `PATCH` will get back a `HTTP 200 OK` response. A representation of the updated resource will be returned in the response.

Deleting a record

Send a `DELETE` request to the detail endpoint.

A successful `DELETE` will get back a `HTTP 204 NO CONTENT` response.

Facility units

A facility may contain within it multiple semi-independent units e.g a pharmacy, a lab and a radiology unit.

Note: These units may fall under the scope of different regulators. This API server does not currently handle that.

Listing multiple records

Facility units can be listed via a `GET` to `/api/facilities/facility_units/`.

In addition to the common filters, facility units can be filtered by the following fields:

Field	Explanation
facility	The `id` of the facility, as listed at `/api/facilities/facilities/` e.g `/api/facilities/facility_units/?facility=faaefb75-dba4-4564-8acb-6b947685de24`
name	The name of the facility unit e.g `/api/facilities/facility_units/?facility=faaefb75-dba4-4564-8acb-6b947685de24&name=pharmacy`
description	The description of the facility unit e.g `/api/facilities/facility_units/?facility=faaefb75-dba4-4564-8acb-6b947685de24&description=hospital%20pharmacy`

Retrieving a single record

A single facility unit record can be retrieved at its detail endpoint i.e `/api/facilities/facility_units/<id>/` e.g `/api/facilities/facility_units/1fcc5c30-9170-4c9d-8d05-9695ba81a08c/`.

Adding a new record

When adding a new facility unit, the fields of interest are the `name`, `description` and `facility`.

The following is a valid `POST` payload for `/api/facilities/facility_units/`:

```
{
    "name": "Pharmacy",
    "description": "Hospital Pharmacy",
    "facility": "faaefb75-dba4-4564-8acb-6b947685de24"
}
```

A successful `POST` will get back a `HTTP 201 Created` response. A representation of the freshly created resource will be returned in the response.

Updating an existing record

A `PATCH` to the detail endpoint will update the relevant field(s): For example:

```
{
    "description": "Community Pharmacy"
}
```

A successful `PATCH` will get back a `HTTP 200 OK` response. A representation of the updated resource will be returned in the response.

Deleting a record

Send a `DELETE` request to the detail endpoint.

A successful `DELETE` will get back a `HTTP 204 NO CONTENT` response.

Facility services

These APIs link facilities to *The service catalog*.

Listing multiple records

The currently registered facility services can be listed via `GET` to `/api/facilities/facility_services/`.

In addition to the standard filters, facility services have the following additional filters:

Field	Explanation
facility	`id` of a facility, as obtained from `/api/facilities/facilities/`
selected_option	`id` of a service catalog service option, as obtained from `/api/facilities/service_options/`

Retrieving a single record

A facility service record can be retrieved at `/api/facilities/facility_services/<id>/` e.g `/api/facilities/facility_services/df6bc639-1d9b-49f8-8f95-51e6de9c93e2/` for the facility service whose `id` is `df6bc639-1d9b-49f8-8f95-51e6de9c93e2`.

Adding a new record

To associate a facility with a service, the required fields are `facility` and `selected_option`.

The following is an example `POST` payload:

```
{
    "service": "f465cb89-995c-4004-9f32-1d97fa6d0eb2",
    "option": "f465cb89-995c-4004-9f32-1d97fa6d0eb2"
}
```

A successful `POST` will get back a `HTTP 201 Created` response. A representation of the freshly created resource will be returned in the response.

Updating an existing record

Issue a `PATCH` to the detail endpoint with the new value. For example, to change the option the example record we created above, the following payload could be sent via `PATCH` to `/api/facilities/facility_services/df6bc639-1d9b-49f8-8f95-51e6de9c93e2/`:

```
{
    "option": "7dde4be8-1c1e-43ce-8569-eebb63bcb329"
}
```

A successful `PATCH` will get back a `HTTP 200 OK` response. A representation of the updated resource will be returned in the response.

Deleting a record

Issue a `DELETE` to the detail endpoint.

A successful `DELETE` will get back a `HTTP 204 NO CONTENT` response.

Facility workflows

Note: These five workflows are the day-to-day operations performed on the MFL system. A well behaved front-end should integrate them into the facility information screens that handle the facility information services mentioned above, rather than give each of these its own set of screens.

Note: These workflows have multiple interactions with the role based access control setup.

The facility "publishing" workflow

The first generation MFL system had a notion of "synchronizing" facility records to the "public site". This notion arose beceause the "public" MFL system was a separate system.

This API does away with that notion. All applications - admin or public, web or mobile - share the same API. Facilities that should be seen in the public API have `is_published` set to `true` and `is_classified` set to `false`.

Note: When `is_classified` is `true`, a user accessing the public site will need to be logged in with an account that has a the `view_classified_facilities` permission.

To "publish" a facility, simply `PATCH` the facility's detail URL and set `is_published` to `true`. Newly created facilities are not published by default.

To "classify" a facility, `PATCH` its detail endpoint with `is_classified` set to `true`. A facility is not classified by default.

Note: The public user interface should add an `is_published=true` filter to every request made to the facilities endpoints. For an unauthenticated user, it should also append `is_classified=false` to every call to the facilities list endpoint.

The administration user interface should implement role based access control, limiting publishing to users with the `publish_facilities` permission.

Facility ratings

Ratings are scores given to a facility's services. One facility service can be rated by multiple users. One user, can rate multiple facility services.

The scores given to a service range from 1 to 5, with 1 being the lowest score and 5 being the highest score.

Note: The facility ratings APIs will be used by both the public and administration user interfaces. The public interface's concern is to facilitate ratings by the general public. The admninistration interface will present read-only summary information.

Facility service ratings

To rate a facility service, simply make a `POST` to `api/facilities/facility_service_ratings/` with the facility_service's `id` and the score given. For example,

```
{
    "facility_service": "80613650-f765-4032-a9d3-bb0fc9cc37cc",
    "rating": 3
}
```

Facility rating reports

The rating reports available include:

1. number of users with specific rating/score

2. sorting of facilities by average score
3. sorting of facility services by average score
4. sorting of facility services in a facility by average score

Facility downloads

Note: Some of these downloads e.g the facility correction template are there for historical reasons. A better approach would involve the use of mobile interfaces (supported by this server's APIs) to facilitate data collection and data updates in the field.

Facility cover letters

To **download** a facilities cover letter:

`GET` the URL `api/facilities/facility_cover_report/<facility_id>/`

Facility correction templates

To **download** a facility's correction template:

`GET` the URL `api/facilities/facility_correction_template/<id>/`

Facilty Inspection Report

To **download** a facility's inspection report: `GET` the URL `api/facilities/facility_inspection_report/<facility_id>/`

Facility Excel reports

Note: The authors of this API treated Excel and CSV output as simply **one more format that data can be exported into**. Excel and CSV data comes from the same serializers that produce the standard API JSON and XML output. This has two positive effects:

- it can use all the available filters
- every list API endpoint (not just the facilities list API endpoint) can produce CSV and Excel

e.g. to get an excel file of facilities: `GET` the URL `api/facilities/facilities/?format=excel`

Facility dashboard APIs

This API is accesbile by administrators at both the county and the National level. The dashboard API does a high level analysis of different aspects of the facilities. Currently, it supports analysis of facilities by owners, administrative units, facility types, facility owner types and analysis by time created.

To get the analyzed data: `GET` the URL `api/facilities/dashboard/`. The data that the endpoint responds with is dependent upon the priviledges of the user logged in.

For a **National user**, the following response is expected


```
{
    "owners_summary": [
        {
            "count": 5,
            "name": "State Coorporation"
        },
        {
            "count": 1203,
            "name": "Private Enterprise (Institution)"
        },
        {
            "count": 0,
            "name": "NOT IN LIST"
        },
        {
            "count": 3,
            "name": "Humanitarian Agencies"
        },
        {
            "count": 179,
            "name": "Private Practice - Unspecified"
        },

    ],
    "recently_created": 8361,
    "county_summary": [
        {
            "count": 784,
            "name": "NAIROBI"
        },
        {
            "count": 392,
            "name": "MERU"
        },
        {
            "count": 379,
            "name": "NAKURU"
        },
        {
            "count": 363,
            "name": "KITUI"
        },
        {
            "count": 358,
            "name": "NYERI"
        },
        {
            "count": 333,
            "name": "KIAMBU"
        },
        {
            "count": 267,
            "name": "KAJIADO"
        },
        {
            "count": 256,
            "name": "MOMBASA"
```

```
        },
        {
            "count": 243,
            "name": "MACHAKOS"
        },
        {
            "count": 233,
            "name": "KILIFI"
        }
    ],
    "total_facilities": 8361,
    "status_summary": [
        {
            "count": 0,
            "name": "Facility_Gazette_By_ID"
        },
        {
            "count": 0,
            "name": "PENDING"
        },
        {
            "count": 0,
            "name": "Not-Operational"
        },
        {
            "count": 8361,
            "name": "OPERATIONAL"
        },
        {
            "count": 0,
            "name": "Licensing"
        },
        {
            "count": 0,
            "name": "Registration"
        },
        {
            "count": 0,
            "name": "Gazettment"
        }
    ],
    "owner_types": [
        {
            "count": 0,
            "name": "Other"
        },
        {
            "count": 268,
            "name": "Non-Governmental Organizations"
        },
        {
            "count": 3226,
            "name": "Private Institutions and Private Practice"
        },
        {
            "count": 853,
            "name": "Faith Based Organization"
        },
```

```
        {
            "count": 356,
            "name": "Other Public Institution"
        },
        {
            "count": 3658,
            "name": "Ministry of Health"
        }
    ],
    "constituencies_summary": [],
    "types_summary": [
        {
            "count": 119,
            "name": "District Hospital"
        },
        {
            "count": 901,
            "name": "Health Centre"
        },
        {
            "count": 3808,
            "name": "Dispensary"
        }
    ]
}
```


For a **County user** (Mombasa County in this case), the following response is expected


```
{
    "owners_summary": [

        {
            "count": 5,
            "name": "Local Authority T Fund"
        },
        {
            "count": 33,
            "name": "Community Development Fund"
        },
        {
            "count": 78,
            "name": "Company Medical Service"
        },
        {
            "count": 265,
            "name": "Non-Governmental Organizations"
        },
        {
            "count": 225,
            "name": "Other Faith Based"
        },
        {
            "count": 10,
            "name": "Supreme Council for Kenya Muslims"
        },

    ],
    "owner_types": [
```

```
        {
            "count": 189,
            "name": "Private Institutions and Private Practice"
        },
        {
            "count": 10,
            "name": "Faith Based Organization"
        },
        {
            "count": 27,
            "name": "Other Public Institution"
        },
        {
            "count": 23,
            "name": "Ministry of Health"
        }
    ],
    "constituencies_summary": [
        {
            "count": 71,
            "name": "MVITA"
        },
        {
            "count": 49,
            "name": "LIKONI"
        },
        {
            "count": 46,
            "name": "NYALI"
        },
        {
            "count": 44,
            "name": "CHANGAMWE"
        },
        {
            "count": 25,
            "name": "KISAUNI"
        },
        {
            "count": 21,
            "name": "JOMVU"
        }
    ],
    "types_summary": [
        {
            "count": 2735,
            "name": "Medical Clinic"
        },
        {
            "count": 196,
            "name": "Other Hospital"
        },
        {
            "count": 119,
            "name": "Sub-District Hospital"
        },
        {
            "count": 172,
```

```
            "name": "Nursing Home"
        }

    ]
}
```

Facility types

There are many types of facilities ranging from health centers, hospitals, dispensaries, national hospitals etc.

Facility types form the basis of upgrading and downgrading of facilities.

A facility type has five distinct fields:

Field	Explanation
id	The primary key of the facility type
name	The name of the facility type e.g HEALTH_CENTER
sub-division	A sub-division of the facility type e.g A hospitla has got several sub divisions e.g District Hospital of Provincial Hospital
preceed-ing	A facility type that comes before the type e.g a Provincial Hospital comes before a National Hospital

Creating A facility type

`POST` to `api/facilities/facility_types/` a payload similar to the one below

```
{
    "name": "Hospital",
    "sub_division": "Provincial Hospital",
    "preceding": "950047f7-dae4-4803-9818-9886004daaf1"
}
```

Expected Response Code: `HTTP 201 CREATED`

Expected sample data:

```
{
    "id": "11494347-f40c-4fbb-8632-cc1f35fe1fc9",
    "created": "2015-05-21T14:38:03.298142Z",
    "updated": "2015-05-21T14:38:03.298162Z",
    "deleted": false,
    "active": true,
    "search": null,
    "name": "Hospital",
    "sub_division": "Provincial Hospital",
    "created_by": 1,
    "updated_by": 1,
    "preceding": "950047f7-dae4-4803-9818-9886004daaf1"
}
```

Listing Facillity types

`GET` the URL `api/facilities/facility_types/`

Sample Response data:


```
    {
        "count": 27,
        "next": "http://localhost:8000/api/facilities/facility_types/?page=2",
        "previous": null,
        "results": [
            {
                "id": "11494347-f40c-4fbb-8632-cc1f35fe1fc9",
                "created": "2015-05-21T14:38:03.298142Z",
                "updated": "2015-05-21T14:38:03.298162Z",
                "deleted": false,
                "active": true,
                "search": null,
                "name": "Hospital",
                "sub_division": "Provincial Hospital",
                "created_by": 1,
                "updated_by": 1,
                "preceding": "950047f7-dae4-4803-9818-9886004daaf1"
            },
            {
                "id": "950047f7-dae4-4803-9818-9886004daaf1",
                "created": "2015-05-15T13:45:13.592372Z",
                "updated": "2015-05-15T13:45:13.592404Z",
                "deleted": false,
                "active": true,
                "search": null,
                "name": "District Hospital",
                "sub_division": null,
                "created_by": 1,
                "updated_by": 1,
                "preceding": null
            }
        ]
}
```


Expected Response code: `HTTP 200 OK`

Retrieving a facility type

`GET` the URL `api/facilities/facility_types/<id>/`

For example to get the details of a facility type whose is `950047f7-dae4-4803-9818-9886004daaf1` do a `GET` to the URL `api/facilities/facility_types/950047f7-dae4-4803-9818-9886004daaf1/`

Sample Response data:


```
{
    "id": "950047f7-dae4-4803-9818-9886004daaf1",
    "created": "2015-05-15T13:45:13.592372Z",
    "updated": "2015-05-15T13:45:13.592404Z",
    "deleted": false,
    "active": true,
    "search": null,
    "name": "District Hospital",
    "sub_division": null,
    "created_by": 1,
    "updated_by": 1,
```

```
    "preceding": null
}
```

Expected Response code `HTTP 200 OK`

Updating Facility types

`PATCH` the URL `api/facilities/facility_types/<id>/` with a payload containing the fields to be edited. For example to update a facility type's name whose id is 950047f7-dae4-4803-9818-9886004daaf1 do a `PATCH` to the URL `api/facilities/facility_types/950047f7-dae4-4803-9818-9886004daaf1/` with a payload similar to the one below

```
{
    "name": "District Hospital Edited"
}
```

Sample Expected Response data:

```
{
    "id": "950047f7-dae4-4803-9818-9886004daaf1",
    "created": "2015-05-15T13:45:13.592372Z",
    "updated": "2015-05-15T13:45:13.592404Z",
    "deleted": false,
    "active": true,
    "search": null,
    "name": "District Hospital Edited",
    "sub_division": null,
    "created_by": 1,
    "updated_by": 1,
    "preceding": null,
}
```

Expected Response Code: `HTTP 200 OK`

Facility Upgrades and Downgrades

Upgrading or downgrading a facility is as easy as changing the facility type of a facility to another type. The person doing this should have the sufficient permssions to do so. This is however a two step process. The First step involves making the upgrade or the downgrade and the second involves confirming the upgrade or the downgrade.

Upgrading/Downgrading a Facility (First Step)

`POST` to `api/facilities/facility_upgrade/` a payload similar to the one shown below

```
{
    "reason": "A good reason for the upgrade",
    "facility": "cc585b49-dc42-47a3-a08a-7f2c39633393", // id of the facility
    "facility_type": "57a0351b-accd-4ccf-b19f-38920ea78e75" // id of the facility type
}
```

Sample Response Data:

```
{
    "id": "70610b2b-ddd8-49b4-8594-52c236a834d2",
    "created": "2015-05-21T15:37:56.240505Z",
    "updated": "2015-05-21T15:37:56.240522Z",
    "deleted": false,
    "active": true,
    "search": null,
    "reason": "A good reason for the upgrade",
    "is_confirmed": false,
    "is_cancelled": false,
    "created_by": 3,
    "updated_by": 3,
    "facility": "cc585b49-dc42-47a3-a08a-7f2c39633393",
    "facility_type": "57a0351b-accd-4ccf-b19f-38920ea78e75"
}
```

Expected Response Code: `HTTP 201 CREATED`

Confirming Upgrade or Downgrade (Second Step)

The CHRIO may choose to either to confirm or cancel a facility upgrade or downgrade.

To confirm a facility upgrade/downgrade `PATCH api/facilities/facility_upgrade/<id>/` where the `id` identifies a particular facility upgrade/downgrade.

For example to confirm the facility upgrade done above do a `PATCH` to `api/facilities/facility_upgrade/70610b2b-ddd8-49b4-8594-52c236a834d2` with the payload below:

```
{
    "is_confirmed": true
}
```

Expected Response data:

```
{
    "id": "70610b2b-ddd8-49b4-8594-52c236a834d2",
    "created": "2015-05-21T15:37:56.240505Z",
    "updated": "2015-05-21T15:37:56.240522Z",
    "deleted": false,
    "active": true,
    "search": null,
    "reason": "A good reason for the upgrade",
    "is_confirmed": true,
    "is_cancelled": false,
    "created_by": 3,
    "updated_by": 3,
    "facility": "cc585b49-dc42-47a3-a08a-7f2c39633393",
    "facility_type": "57a0351b-accd-4ccf-b19f-38920ea78e75"
}
```

Expected HTTP Response code `HTTP 200 OK`

Cancelling a facility upgrade/downgrade(Second Step)

Cancelling a facility upgrade or downgrade is very similar to confirming a facility upgrade with a minor change in the payload sent.

Do a PATCH to the url api/facilities/facility_upgrade/<id>/ with a payload similar to the one shown below:

```
{
    "is_cancelled": true
}
```

Note: It is after the confirmation of a facility upgrade or downgrade that a facility is deemed to have been upgraded or downgraded and the changes reflected in the facility.

Listing Facilities that are due for upgrade/downgrade Confirmation

GET the URL /api/facilities/facility_upgrade/?is_confirmed=false

Sample Response data:

```
    {
        "count": 1,
        "next": null,
        "previous": null,
        "results": [
            {
                "id": "70610b2b-ddd8-49b4-8594-52c236a834d2",
                "created": "2015-05-21T15:37:56.240505Z",
                "updated": "2015-05-21T15:37:56.240522Z",
                "deleted": false,
                "active": true,
                "search": null,
                "reason": "A good reason for the upgrade",
                "is_confirmed": false,
                "is_cancelled": false,
                "created_by": 3,
                "updated_by": 3,
                "facility": "cc585b49-dc42-47a3-a08a-7f2c39633393",
                "facility_type": "57a0351b-accd-4ccf-b19f-38920ea78e75"
            }
    ]
}
```

Expected Response code: HTTP 200 OK

Listing all the facilties whose upgrades and downgrades have been declined: GET the URL /api/facilities/facility_upgrade/?is_cacelled=true

The resulting payload and expected response code are similar the ones above

Listing all the the Upgrades/Downgrades of a facility

GET the URL /api/facilities/facility_upgrade/?facility=<id>

For example a get to the URL api/facilities/facility_upgrade/?facility=cc585b49-dc42-47a3-a08a-7f2c3963339311 results in the data shown below and the a response code of HTTTP 200 OK

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": "70610b2b-ddd8-49b4-8594-52c236a834d2",
            "created": "2015-05-21T15:37:56.240505Z",
            "updated": "2015-05-21T15:37:56.240522Z",
            "deleted": false,
            "active": true,
            "search": null,
            "reason": "A good reason for the upgrade",
            "is_confirmed": true,
            "is_cancelled": true,
            "created_by": 3,
            "updated_by": 3,
            "facility": "cc585b49-dc42-47a3-a08a-7f2c39633393",
            "facility_type": "57a0351b-accd-4ccf-b19f-38920ea78e75"
        }
    ]
}
```


Community Health Units

This chapter assumes that the reader is familiar with the general principles explained in the *Using the API - basic principles* chapter.

Kenya's community health strategy relies on community health workers for outreach at the lowest levels (embedded into communities). These workers are organized into community health units. The second edition of the Master Facilities List provides APIs for the management of community health units.

This chapter concerns itself with the resources that model community health units and link them to facilities.

Note: Community health units are an extension of the Master Facilities List. A community health unit is a health service delivery structure within a defined geographic area covering a population of approximately 5,000 people.

Each unit is assigned 2 Community Health Extension Workers (CHEWs) and community health volunteers who offer promotie, preventative and basic curative services.

Each unit is governed by a Community Health Committee (CHC) and is **linked to a specific health facility**.

The role of a community health unit is to bring services closer to the people that need them. Those services include:

- Water and sanitation hygiene; e.g. Faecal management, Household water treatment and demonstrations on hand washing with soap, etc.
- Advice on maternal and child health e.g. Immunization, Individual birth plan, etc.
- Provision of Family planning commodities.
- Growth monitoring for children under 5 years.
- Deworming of children.
- Provision of Long Lasting Insecticide Treated Nets (LLITNs).
- Management of diarrhea, injuries, wounds, jiggers and other minor illnesses.

- Provision of Information, Education & Communication (IEC) materials
- Defaulter tracing (ART, TB and Immunization)
- Referrals to health facilities
- First Aid Services

The implementation of community health units in this API is semi-independent. The units connect to the rest of MFL at only one point - their linkage to facilities.

Community Health Unit Approvers

The community health approvers resource holds the details of entities that are involved in approval of community health units.

The known approvers can be listed by issuing a `GET` to `/api/chul/approvers/`. To register a new approving entity, you need to supply a `name`, `description` and `abbreviation`. The following example illustrates that:

```
{
    "name": "Division of Community Health Services",
    "description": "Division of Community Health Services, Ministry of Health",
    "abbreviation": "DCHS"
}
```

Community Health Unit Statuses

The community health unit statuses that are known / available can be listed at `/api/chul/statuses/` via `GET`. These will be used to mark the current status of a community health unit, and when analysing the status of registered community health units.

To create a new status, you need to `POST` a `name` and a `description`. Here is an example payload:

```
{
    "name": "ACTIVE",
    "description": "Actively Deployed"
}
```

Note: This reflects the operational status of the Community Health Unit.

Community Health Units

Community health units can be listed via `GET` to `/api/chul/units/`.

To add a new community health unit, `POST` to `/api/chul/units/`, `POST` a payload that has a `name`, `facility` and `status`. For the facility and status, the `id` s are sent (foreign keys).

For example:

```
{
    "name": "Gachie Health Unit",
    "facility": "2927d31f-b1a0-4d17-93b0-ea648af7b9f0",
    "status": "0e2ba3fc-9c81-4c30-b52e-b62664462cb7"
}
```

Note: The community health unit `code` is auto-assigned. Immediately after creating the facility record, the code (and other auto-assigned fields) will be inserted in the response.

Community Health Unit Contacts

A community health unit may be linked to zero or more contacts. The contacts will have been created at `/api/common/contacts/` using APIs that are discussed in the *Metadata resources* chapter.

Community health unit contacts can be listed and created at `/api/common/contacts/`. To list a community health unit to a contact, `POST` to that endpoint the `id` of the contact and the `id` of the community health unit. The example payload below illustrates that:

```
{
    "health_unit": "2d425ab7-0002-4b95-9cd1-638972efb75d",
    "contact": "7dd62ab9-94c2-48d6-a10f-d903bd57acd5"
}
```

Community Health Unit Approvals

The approval status of community health units is listed / maintained at `/api/chul/unit_approvals/`.

To record a new approval, you should supply a `comment`, `approval_date`, `approver`, `approval_status` and `health_unit`.

The `approver` is the `id` of an approver registered at `/api/chul/approvers/`. The `approval_status` is the `id` of an approval status registered at `/api/chul/approval_statuses/`. The `health_unit` is the `id` of a community health unit registered at `/api/chul/units/`. The `comment` is a free-text explanation, while the `approval_date` is an ISO 8601 **date** (not datetime) string that represents the date when the approval occured.

The following example is a valid `POST` payload:

```
{
    "comment": "For documentation / training purposes",
    "approval_date": "2015-05-09",
    "approver": "02b610c1-067f-4e0c-9bad-31cc029f6ee3",
    "approval_status": "44c2abfd-3944-484f-ae4c-b30778e25398",
    "health_unit": "96645d26-8e4e-4078-9e10-a5176f5432df"
}
```

Note: This reflects the approval status of the Community Health Unit.

Community Health Workers

Community health workers are attached to community health units. They are listed and maintained at `/api/chul/workers/`.

When registering a new community health worker, supply a `first_name`, `last_name`, `surname`, `id_number` and `health_unit`. The `health_unit` is the `id` of the community health unit that the worker is attached to, and can be retrieved from `/api/chul/units/`.

```
{
    "first_name": "Does",
    "last_name": "Not",
    "surname": "Exist",
    "id_number": 545432,
    "health_unit": "96645d26-8e4e-4078-9e10-a5176f5432df"
}
```

Community Health Workers Contacts

A community health worker can be linked to a contact that has already been registered at `/api/common/contacts/` by `POST` ing to `/api/chul/workers_contacts/` the `id` of the worker and the `id` of the contact.

For example:

```
{
    "health_worker": "db04b653-b0f7-434f-a224-3ea4d93b69c1",
    "contact": "2d04afdc-46a8-4b11-85b8-63f5c035366f"
}
```

Community Health Workers Approvals

The approval status of community health workers is maintained at `/api/chul/worker_approvals/`.

The key pieces of information to maintain about each approval are the `approver` (an `id` of an approver registered at `/api/chul/approvers/`), `approval_status` (`id` of an approval status registered at `/api/chul/approval_statuses/`) and `health_worker` (`id` of a health worker registered at `/api/chul/workers/`) and a free-form `comment`.

The example below is a valid `POST` payload:

```
{
    "approver": "02b610c1-067f-4e0c-9bad-31cc029f6ee3",
    "approval_status": "44c2abfd-3944-484f-ae4c-b30778e25398",
    "health_worker": "db04b653-b0f7-434f-a224-3ea4d93b69c1",
    "comment": "Documentation example"
}
```

Regulation

This chapter assumes that the reader is familiar with the general principles explained in the *Using the API - basic principles* chapter.

Every healthcare facility falls under the regulatory scope of at least one regulator. For example - at the time of writing, most healthcare facilities are licensed by the Kenya Medical Practitioners and Dentists Board.

Regulators have their own information systems. The MFL provides APIs that can facilitate two way data flow between the regulators' systems and the Master Facilities List.

For regulation of facilities to occur in the system two entities are required:

1. The regulating body
2. The regulation status

Regulatory Bodies

These are the bodies that are in-charge of assessing whether a facility should be licensed, gazetted or registered. They also determine the KEPH level of operation of a facility.

Creation

`POST` to `/api/facilities/regulating_bodies/` a payload similar to the one shown below:

```
{
    "name": "Kenya Medical Practitioners Pharmacists  and Dentists Board",
    "abbreviation": "KMPPDB",
    "regulation_verb": "license", // e .g gazette license register
    "regulatory_body_type": "d195219b-7b5b-4395-889b-3dbcb7bfccf6" // this is the id␣
↪of the owner type of facilities they regulate
}
```

Expected Response code `HTTP 201 CREATED`

Sample response data:

```
{
    "id": "fbb96308-454f-4d1d-9ca4-597018d460b7",
    "created": "2015-05-08T16:24:09.552222Z",
    "updated": "2015-05-08T16:24:09.552245Z",
    "deleted": false,
    "active": true,
    "search": null,
    "name": "Kenya Medical Practitioners Pharmacists  and Dentists Board",
    "abbreviation": "KMPDB",
    "regulation_verb": "license",
    "created_by": 3,
    "updated_by": 3,
    "regulatory_body_type": null,
    "contacts": []
}
```

Updating

Do a `PATCH` to `/api/facilities/regulating_bodies/<id>` with a payload containing only the fields that are to be modified.

For example:

```
{
    "name": "Kenya Medical Practitioners Pharmacists  and Dentists Board"
}
```

Expected HTTP Response code `HTTP 200 OK`

Sample response data:

```
{
    "id": "fbb96308-454f-4d1d-9ca4-597018d460b7",
    "created": "2015-05-08T16:24:09.552222Z",
    "updated": "2015-05-08T16:24:09.552245Z",
```

```
    "deleted": false,
    "active": true,
    "search": null,
    "name": "Kenya Medical Practitioners Pharmacists  and Dentists Board edited",
    "abbreviation": "KMPPDB",
    "regulation_verb": "license",
    "created_by": 3,
    "updated_by": 3,
    "regulatory_body_type": null,
    "contacts": []
}
```


Listing

Do a `GET` the `/api/facilities/regulating_bodies/`

Below is a sample response data from the endpoint:


```
{
    "count": 2,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": "bdc6d243-af73-438f-be01-224f621bf538",
            "created": "2015-05-08T15:58:18.351751Z",
            "updated": "2015-05-08T15:58:18.351772Z",
            "deleted": false,
            "active": true,
            "search": null,
            "name": "Pharmacy & Poisons Board",
            "abbreviation": "Pharmacy & Poisons Board",
            "regulation_verb": "Licensing",
            "created_by": 1,
            "updated_by": 1,
            "regulatory_body_type": null,
            "contacts": []
        },
        {
            "id": "5a797ac9-dbbb-4579-b2c3-dee80c2ae43b",
            "created": "2015-05-08T15:58:18.346141Z",
            "updated": "2015-05-08T15:58:18.346164Z",
            "deleted": false,
            "active": true,
            "search": null,
            "name": "Clinical Officers Council",
            "abbreviation": "COC",
            "regulation_verb": "Licensing",
            "created_by": 1,
            "updated_by": 1,
            "regulatory_body_type": null,
            "contacts": []
        }
    ]
}
```

Retrieving

To retrieve a single regulatory body do a `GET` to the url `api/facilities/regulating_bodies/<id>/` Id being the id of the regulatory body. The response data will be similar to the data shown below:


```
{
    "id": "bdc6d243-af73-438f-be01-224f621bf538",
    "created": "2015-05-08T15:58:18.351751Z",
    "updated": "2015-05-08T15:58:18.351772Z",
    "deleted": false,
    "active": true,
    "search": null,
    "name": "Pharmacy & Poisons Board",
    "abbreviation": "Pharmacy & Poisons Board",
    "regulation_verb": "Licensing",
    "created_by": 1,
    "updated_by": 1,
    "regulatory_body_type": null,
    "contacts": []
}
```


Regulatory Statuses

A regulation state is a state in which the facility will be after the regulator has assessed a facility's suitability for that state.

The default states are as provided in the implementation guide.

1. PENDING_LICENSING
2. LICENSED
3. LICENSE_SUSPENDED
4. LICENSE_CANCELLED
5. PENDING_REGISTRATION
6. REGISTERED
7. PENDING_GAZETTEMENT
8. GAZETTED

Listing

Do a `GET` to the url `api/facilities/regulation_status/` Example response


```
{
    "count": 2,
    "next": null,
    "previous": null,
    "results": [
        {
            "id": "d195219b-7b5b-4395-889b-3dbcb7bfccf6",
            "next_state_name": "",
            "previous_state_name": "Pending Registration",
            "created": "2015-05-08T10:00:48.608555Z",
```

```
            "updated": "2015-05-08T10:00:48.608572Z",
            "deleted": false,
            "active": true,
            "search": null,
            "name": "Registered",
            "description": null,
            "is_initial_state": false,
            "is_final_state": false,
            "created_by": 1,
            "updated_by": 1,
            "previous_status": "1390d5c3-9226-44a0-b464-13d17fed2b41",
            "next_status": null
        },
        {
            "id": "5287dbfc-e2c0-410f-80e3-7ec20ac4dc79",
            "next_state_name": "",
            "previous_state_name": "Pending Gazettment",
            "created": "2015-05-08T10:00:48.601773Z",
            "updated": "2015-05-08T10:00:48.601808Z",
            "deleted": false,
            "active": true,
            "search": null,
            "name": "Gazettment",
            "description": null,
            "is_initial_state": false,
            "is_final_state": true,
            "created_by": 1,
            "updated_by": 1,
            "previous_status": "06d215ec-4a8c-469f-88df-028e597a348d",
            "next_status": null
        }
    ]
}
```


Creation

Creating a regulation status requires one to know the entire regulation workflow of a facility from the first state to the last state. This is so since as one configures a state they have to know whether it is the initial state, the final state or an intermediary state.

This section will be divided into 3 parts.

1. Creating an initial state

To create the very first regulation state. To create it do a `POST` to the `api/facilities/regulation_status/` with the similar to the one shown below.

```
{
    "name": "PENDING_LICENSING",
    "description": "This is the very first state after a facility has been approved
↪by the CHRIO",
    "is_initial_state": true,

}
```

Expected response code. `HTTP 201 CREATED`

Sample Response data:

```
{
    "id": "698e1e45-0ab7-466f-a449-9091036cfa31",
    "next_state_name": "Pending Licensing",
    "previous_state_name": "Pending Licensing",
    "created": "2015-05-08T16:17:32.016528Z",
    "updated": "2015-05-08T16:17:32.016543Z",
    "deleted": false,
    "active": true,
    "search": null,
    "name": "PENDING_LICENSING",
    "description": "This is the very first state after a facility has been approved␣
↪by the CHRIO",
    "is_initial_state": false,
    "is_final_state": true,
    "created_by": 3,
    "updated_by": 3,
    "previous_status": "1938861f-2c34-49c5-808f-caa0ed1c3681",
    "next_status": "1938861f-2c34-49c5-808f-caa0ed1c3681"
}
```

2. Creating a final State

Creating a final state is very similar to creating an initial state.

`POST` to `/api/facilities/regulation_status/` The only change will be to substitute the is_initial_state with is_final_state and add a previous_state to the sample payload.

```
{
    "name": "LICENSED",
    "description": "This is the final state after a  facility has been given a␣
↪license by the regulating body",
    "is_final_state": true,
    'previous_state': "1938861f-2c34-49c5-808f-caa0ed1c3681" // id of the preceding␣
↪state
}
```

Expected response code: `HTTP 201 CREATED`

Sample Response data:

```
{
    "id": "698e1e45-0ab7-466f-a449-9091036cfa31",
    "next_state_name": "Pending Licensing",
    "previous_state_name": "Pending Licensing",
    "created": "2015-05-08T16:17:32.016528Z",
    "updated": "2015-05-08T16:17:32.016543Z",
    "deleted": false,
    "active": true,
    "search": null,
    "name": "LICENSED",
    "description": "This is the final state after a  facility has been given a␣
↪license by the regulating body",
    "is_initial_state": false,
    "is_final_state": true,
```

```
    "created_by": 3,
    "updated_by": 3,
    "previous_status": "1938861f-2c34-49c5-808f-caa0ed1c3681",
    "next_status": "1938861f-2c34-49c5-808f-caa0ed1c3681"
}
```

3. Creating an intermediary State.

An intermediary should have a preceding and succeeding state. Here is an example:

`POST` to `/api/facilities/regulation_status/`

```
{
    "name": "INTERMEDIARY_STATE",
    "description": "This is the state in-between state 1 and state 3",
    "previous_status": "1938861f-2c34-49c5-808f-caa0ed1c3681" // id of the preceding␣
↪state ,
    "next_status": "1938861f-2c34-49c5-808f-caa0ed1c3681" // id of the suceeding state
}
```

Expected response `HTTP 201 CREATED`

sample Reponse data:

```
{
    "id": "698e1e45-0ab7-466f-a449-9091036cfa31",
    "next_state_name": "Pending Licensing",
    "previous_state_name": "Pending Licensing",
    "created": "2015-05-08T16:17:32.016528Z",
    "updated": "2015-05-08T16:17:32.016543Z",
    "deleted": false,
    "active": true,
    "search": null,
    "name": "INTERMEDIARY_STATE",
    "description": "This is the state in-between state 1 and state 3",
    "is_initial_state": false,
    "is_final_state": false,
    "created_by": 3,
    "updated_by": 3,
    "previous_status": "1938861f-2c34-49c5-808f-caa0ed1c3681",
    "next_status": "1938861f-2c34-49c5-808f-caa0ed1c3681"
}
```

Retrieving a single regulatory state

Do a `GET` to the url `/api/facilities/regulation_status/<id>` where id is the id of the regulatory state.

```
{
    "id": "d195219b-7b5b-4395-889b-3dbcb7bfccf6",
    "next_state_name": "",
    "previous_state_name": "Pending Registration",
    "created": "2015-05-08T10:00:48.608555Z",
    "updated": "2015-05-08T10:00:48.608572Z",
    "deleted": false,
    "active": true,
```

```
    "search": null,
    "name": "Registered",
    "description": null,
    "is_initial_state": false,
    "is_final_state": false,
    "created_by": 1,
    "updated_by": 1,
    "previous_status": "1390d5c3-9226-44a0-b464-13d17fed2b41",
    "next_status": null
}
```


Expected Response `HTTP 200 OK`

Updating a regulatory state

Do a `PATCH` to `/api/facilities/regulation_status/<id>/` with the payload being the fields to update. Here is a sample payload


```
{

    "name": "Registered Edited"
}
```


The above payload will update the details of the state whose id is the url.

Expected Response code: `HTTP 200 OK`

Sample Response data:


```
{
    "id": "d195219b-7b5b-4395-889b-3dbcb7bfccf6",
    "next_state_name": "",
    "previous_state_name": "Pending Registration",
    "created": "2015-05-08T10:00:48.608555Z",
    "updated": "2015-05-08T10:00:48.608572Z",
    "deleted": false,
    "active": true,
    "search": null,
    "name": "Registered Edited",
    "description": null,
    "is_initial_state": false,
    "is_final_state": false,
    "created_by": 1,
    "updated_by": 1,
    "previous_status": "1390d5c3-9226-44a0-b464-13d17fed2b41",
    "next_status": null
}
```


Listing Facilities pending regulation

Do a `GET` to the url `/api/facilities/facility_regulation_status/?regulated=False` This will respond with a list of the facilities that have been modified and need to be regulated or the facilities that have been not yet been regulated. The response data will be similar to the sample response data below:

```
{
"count": 1,
"next": null,
"previous": null,
"results": [
        {
            "id": "8c0964a1-b733-40e4-b0be-1874749e469b",
            "regulary_status_name": null,
            "facility_type_name": "District Hospital",
            "owner_name": "Ministry of Health",
            "owner_type_name": "Ministry of Health",
            "county": "TRANS NZOIA",
            "constituency": "ENDEBESS",
            "created": "2015-05-08T09:58:36.862227Z",
            "updated": "2015-05-08T09:58:36.862242Z",
            "deleted": false,
            "active": true,
            "search": null,
            "name": "Endebess District Hospital",
            "code": 14455,
            "abbreviation": "",
            "description": "",
            "location_desc": "Kitale Swam Road",
            "number_of_beds": 20,
            "number_of_cots": 8,
            "open_whole_day": true,
            "open_whole_week": true,
            "is_classified": false,
            "is_published": true,
            "is_synchronized": false,
            "created_by": 1,
            "updated_by": 1,
            "facility_type": "1d2e7d02-97e0-470b-9889-549df3ff49f8",
            "operation_status": "e865f01b-8937-40fc-a095-fbbb83c59461",
            "ward": "a4223139-30e4-4253-88fe-405a622aa2f7",
            "owner": "7506421d-7838-4eee-9a44-7c92fd76d0b9",
            "officer_in_charge": null,
            "physical_address": "3c75fb20-619d-4591-8f93-56f7493ee764",
            "parent": null,
            "contacts": []
        },
        {
            "id": "854bb94d-7a87-45c7-9243-4b9d9751a690",
            "regulary_status_name": null,
            "facility_type_name": "Health Centre",
            "owner_name": "Ministry of Health",
            "owner_type_name": "Ministry of Health",
            "county": "TRANS NZOIA",
            "constituency": "ENDEBESS",
            "created": "2015-05-08T09:58:36.849294Z",
            "updated": "2015-05-08T09:58:36.849311Z",
            "deleted": false,
            "active": true,
            "search": null,
            "name": "Kwanza Health Centre",
            "code": 15003,
            "abbreviation": "",
            "description": "",
```

```
            "location_desc": "",
            "number_of_beds": 18,
            "number_of_cots": 0,
            "open_whole_day": false,
            "open_whole_week": true,
            "is_classified": false,
            "is_published": true,
            "is_synchronized": false,
            "created_by": 1,
            "updated_by": 1,
            "facility_type": "3c8a65ec-8489-4483-b32b-057098a9fe08",
            "operation_status": "e865f01b-8937-40fc-a095-fbbb83c59461",
            "ward": "4e203a27-8c37-468e-8b39-407193a6d862",
            "owner": "7506421d-7838-4eee-9a44-7c92fd76d0b9",
            "officer_in_charge": null,
            "physical_address": "3c75fb20-619d-4591-8f93-56f7493ee764",
            "parent": null,
            "contacts": []
        }
    ]
}
```


Regulate a facility

`POST` to `/api/facilities/facility_regulation_status/` a payload similar to the one shown below:


```
{

    "reason": "The facility has met all the requirements",
    "license_number": "F135/2015",
    "facility": "d0cf7632-2854-464f-8638-03d1c021f519",
    "regulating_body": "ed3ac8af-c1a7-42f4-9f0d-a9c5e4cf3c13",
    "regulation_status": "5287dbfc-e2c0-410f-80e3-7ec20ac4dc79"
}
```


Expected Reponse Code `HTTP 201 Created`

Sample Reponse data:


```
{
    "id": "594f7bd1-ce6b-4a6d-82c2-523b1710ec31",
    "created": "2015-05-08T16:10:22.604609Z",
    "updated": "2015-05-08T16:10:22.604631Z",
    "deleted": false,
    "active": true,
    "search": null,
    "reason": "The facility has met all the requirements",
    "license_number": "F135/2015",
    "is_confirmed": false,
    "is_cancelled": false,
    "created_by": 3,
    "updated_by": 3,
    "facility": "d0cf7632-2854-464f-8638-03d1c021f519",
    "regulating_body": "ed3ac8af-c1a7-42f4-9f0d-a9c5e4cf3c13",
    "regulation_status": "5287dbfc-e2c0-410f-80e3-7ec20ac4dc79"
}
```

GIS Support

This chapter assumes that the reader is familiar with the general principles explained in the *Using the API - basic principles* chapter.

The MFL 2 API server uses the excellent GeoDjango and PostGIS to provide *The service catalog* that can be used to generate facility maps, perform geographic queries and validate facility coordinate data. You can read more about this at the *GIS Support* page.

What is GIS?

A geographic information system (GIS) lets us visualize, question, analyze, and interpret data to understand relationships, patterns, and trends.

Master Facility List data is inherently geographical - the Master Facility List should have coordinates for all facilities in Kenya. The GIS APIs provided by this server support the visualization, interogation and analysis of this data.

Note: The official front-ends barely scratch the surface when it comes to the use of GIS data. These APIs are open to third party applications too.

GIS data formats

There are many GIS file formats to choose from. We chose to go with GeoJSON because it fits in with our general preference for JSON. It is easy to convert from GeoJSON to ESRI Shapefile and KML formats.

A brief note about points

In "day to day language", we might be accustomed to expressing points as *(latitude, longitude)* pairs e.g *(-1.300462, 36.791533)* for the location of this writer's office at the time of writing. When expressing that location as a GeoJSON "point", we'll need to "flip" the coordinates, so that the GeoJSON for this author's office would be:

```
{
    type: "Point",
    coordinates: [
        36.791533,
        -1.300462
    ]
}
```

How do I move from GeoJSON to a map?

If you are building a web application, take a look at Leaflet and OpenLayers.

If you are working on a mobile application, you could take a look at the Google Maps API or its competitors e.g Bing Maps.

If you are working on on a desktop application, we assume that you know what you are doing and do not need any helpful pointers.

Administrative units

Kenya has a three tier administrative structure: the country has 47 **counties**. Each county has a number of **constituencies**, with the total for the country being 290 constituencies. Each constituency has a number of wards, with the total for the country being 1450 wards. The GIS enabled APIs follow this administrative structure.

Note: This server also has resources that contain country boundaries. The default distribution has data from the *World Borders Dataset* (from http://thematicmapping.org/).

We have not documented the country boundary APIs for the following reasons:

- The county, constituency and ward boundary APIs meet all of the Kenyan MFL needs.
- The borders in the World Borders Dataset are inaccurate - sometimes lopping off several square kilometers around the borders.

Note: The default distribution has map (boundary) data for 1482 out of 1450 wards.

The administrative unit data is considered "setup data" - loaded at server install time, rarely changed afterward. For that reason, the documentation will focus on retrieval and interpretation. If you need to change or add, the basic principles explained in the *Using the API - basic principles* chapter still apply.

Counties

Counties can be listed by sending a `GET` to `/api/common/counties/`. Every county is identified by a `name` and `code`.

An individual county's detail record is available at `/api/common/counties/<pk>/` e.g `/api/common/counties/dd999449-d36b-47f2-a958-1f5bb52951d4/` for the county whose `id` is `dd999449-d36b-47f2-a958-1f5bb52951d4`.

Note: The county detail view is "rich". It embeds a `facility_coordinates` key that shows the location of every facility in that county.

The facility co-ordinates are a map, with the facility names as keys. For example:

```
facility_coordinates: {
    AAR Gwh Health Care Ltd: {
        type: "Point",
        coordinates: [
            36.80897,
            -1.29467
        ]
    },
    Dr Musili Clinic (Afya Centre-Nairobi): {
        type: "Point",
            coordinates: [
                36.82763,
                -1.28799
        ]
    },
    // truncated for brevity
```

The county detail view also embeds within itself the appropriate `county_boundary`. The contents of this will be discussed in the next section.

County Boundaries

County boundaries can be listed at `/api/gis/county_boundaries/`. The list view is a GeoJSON "FeatureCollection", while the detail view is a GeoJSON "Feature".

Note: The border (polygon) is under the `geometry` key for every feature.

Every boundary (feature) serialization has the following fields:

- `center` - a `Point` that represents the **geometric centre** of the area
- `facility_count` - the number of facilities in that geographic area
- `density` - a **synthetic value** (roughly comparable to facilities per square kilometer, although it is not actually facilities / sq.km). This is used by front-end clients to color-code maps.
- `constituency_ids` - a list of the `id` s (primary keys) of the constituencies under that county. These can be appended to the `/api/common/constituencies/` endpoint i.e `/api/constituencies/<id>/` in order to retrieve the details of each constituency in the county.
- `constituency_boundary_ids` - a list of the `id` s of the constituency boundary objects for the constituencies under the county in question. These can be used to retrieve the constituency boundaries at `/api/gis/constituency_boundaries/<pk>/`.

Constituencies

Constituencies can be listed by sending a `GET` to `/api/common/constituencies/`. Every constituency is identified by a `name` and a `code`.

Note: The constituency detail view is, like the county detail view, "rich". It embeds `facility_coordinates` and the relevant `constituency_boundary`.

Constituency Boundaries

Constituency boundaries can be listed at `/api/gis/constituency_boundaries/`. The output is similar to that of the county boundary endpoints, with the following differences: it embeds `ward_ids` instead of `constituency_ids` and `ward_boundary_ids` instead of `constituency_boundary_ids`.

Wards

Wards can be listed by sending a `GET` to `/api/common/wards/`. Every ward is identified by a `name` and a `code`.

Note: The ward detail view is, like the county and constituency detail views, "rich". It embeds `facility_coordinates` and the relevant `ward_boundary`.

Ward Boundaries

Ward boundaries can be listed at `/api/gis/ward_boundaries/`. The output is similar to that of the county boundary endpoints, with the following differences: as the smallest administrative unit, a ward does not embed the coordinates of any other administrative unit.

Lookup administrative units

It is possible to determine the location of a facility using it's coordinates by sending a `POST` to `/api/mfl_gis/ikowapi/` with the *longitude* and *latitude*.

```
{
    "longitude": 1.3213,
    "latitude": 4.53434
}
```

The response, if successful, shall provide the ward, constituency and county ids and names.

```
{
    "ward": "<ward id>",
    "ward_name": "<ward name>",
    "ward_code": "<ward code>",
    "constituency": "<constituency id>",
    "constituency_name": "<constituency name>",
    "constituency_code": "<constituency code>",
    "county": "<id of the county>",
    "county_name": "<county name>",
    "county_code": "<county code>"
}
```

Facility Coordinates

The facility coordinates resources can be found at `/api/gis/coordinates/`. The example below will be used to explain the format:

```
{
    id: "1051cac1-b6e1-46c6-8782-a182dd1a9c50",
    type: "Feature",
    geometry: {
        type: "Point",
        coordinates: [
            34.92687,
            0.88226
        ]
    },
    properties: {
        created: "2015-05-06T17:29:47.710254Z",
        updated: "2015-05-06T17:29:47.710266Z",
        deleted: false,
        active: true,
        search: null,
        collection_date: "2015-05-06T17:29:48.624415Z",
        created_by: 1,
        updated_by: 1,
        facility: "7f91fb27-8fa5-4160-b572-2dc0ad7a554e",
```

```
        source: "c027c6fa-19b2-4fcd-83fa-f84705be84ea",
        method: "1a3f3df8-8c18-4cac-89cc-93dc59a0e057"
    }
}
```

The facility's location is the `geometry Point`. The facility in question is identified by the `facility` property, which contains a facility primary key that can be used to retrieve the facilities from `/api/facilities/facilities/<pk>/` e.g. `/api/facilities/facilities/7f91fb27-8fa5-4160-b572-2dc0ad7a554e/` for the example above.

To set up new facility coordinates, `POST` to `/api/gis/coordinates/` a payload similar to the example below:

```
{
    "coordinates": {
        "type": "Point",
        "coordinates": [
            34.96962,
            0.45577
        ]
    },
    "facility": "be6ca131-5767-45b2-8213-104214becdd3",
    "source": "c027c6fa-19b2-4fcd-83fa-f84705be84ea",
    "method": "cd0bbbcf-60fa-4b76-b48c-5dcda414b43d"
}
```

Every geocode is associated with a geocode source and a geocode method. The `source` key in the payload above is for the geocode source while the `method` key is for the geocode method.

Geocode sources are viewed/created at `/api/gis/geo_code_sources/` while geocode methods are viewed/created at `/api/gis/geo_code_methods/`. Both take a `name` and a `description`.

Workflow

MFL API v2 is a liberally licensed (MIT license) project. All development occurs in the open on the MFL API Github project. We use the MFL API Github issue list to manage bug reports and enhancement requests.

This project uses the GitFlow Workflow.

In summary:

- all work should occur in feature branches
- the target for pull requests is the `develop` branch
- the release manager (presently @ngurenyaga) will periodically create release branches that ultimately get merged into `master` and tagged
- fixes on released versions will occur in hotfix branches

We adhere to semantic versioning - https://semver.org .

In order to deploy a new version, you will need to have a `$HOME/.pypirc` that has the correct pypi credentials. The command to deploy is `fab deploy`. The credentials are not stored on GitHub - for obvious reasons.

Contributors' code of conduct

We have an open-door policy when it comes to contributions. At the same time, we'd like to build a friendly community up around the MFL project, and to be good citizens of the open source clinical informatics landscape.

This project welcomes input - which could be code, but also documentation, training, support on adoption, bug reports and feature requests.

Our code of conduct is based on the Django Code of Conduct. The important parts are reproduced below:

- **Be friendly and patient**.
- **Be welcoming**. We strive to be a community that welcomes and supports people of all backgrounds and identities. This includes, but is not limited to members of any race, ethnicity, culture, national origin, colour, immigration status, social and economic class, educational level, sex, sexual orientation, gender identity and expression, age, size, family status, political belief, religion, and mental and physical ability.
- **Be considerate**. Your work will be used by other people, and you in turn will depend on the work of others. Any decision you take will affect users and colleagues, and you should take those consequences into account when making decisions. Remember that we're a world-wide community, so you might not be communicating in someone else's primary language.
- **Be respectful**. Not all of us will agree all the time, but disagreement is no excuse for poor behavior and poor manners. We might all experience some frustration now and then, but we cannot allow that frustration to turn into a personal attack. It's important to remember that a community where people feel uncomfortable or threatened is not a productive one. Members of the Django community should be respectful when dealing with other members as well as with people outside the Django community.
- **Be careful in the words that you choose**. We are a community of professionals, and we conduct ourselves professionally. Be kind to others. Do not insult or put down other participants. Harassment and other exclusionary behavior aren't acceptable. This includes, but is not limited to:
  - Violent threats or language directed against another person.
  - Discriminatory jokes and language.
  - Posting sexually explicit or violent material.
  - Posting (or threatening to post) other people's personally identifying information ("doxing").
  - Personal insults, especially those using racist or sexist terms.
  - Unwelcome sexual attention.
  - Advocating for, or encouraging, any of the above behavior.
  - Repeated harassment of others. In general, if someone asks you to stop, then stop.
- **When we disagree, try to understand why**. Disagreements, both social and technical, happen all the time and Django is no exception. It is important that we resolve disagreements and differing views constructively. Remember that we're different. The strength of Django comes from its varied community, people from a wide range of backgrounds. Different people have different perspectives on issues. Being unable to understand why someone holds a viewpoint doesn't mean that they're wrong. Don't forget that it is human to err and blaming each other doesn't get us anywhere, rather offer to help resolving issues and to help learn from mistakes.

Regulator Synchronization

Regulator synchronization is divided into two sections:

1. *Part 1: Attached Facilities Synchronization*

2. *Part 2: Stand Alone Facilities Synchronization*

Part 1: Attached Facilities Synchronization

Attached facilities are those facilities that offer specialized health services such as pharmaceutical services, laboratory services, opthalmology services, physiotherapy services etc besides offering other general health services.

This is for **Attached facilities** that are initially created in the RHRIS system and do not have a master facility code assigned to them. This endpoint provides a way to enable the regulatory system to notify the master facility list(MFL) that there are facilities that have been registered and they are not in the MFL. After which the concerned officer (CHRIO) can ensure that the facilities are registered with the MFL.

The synchronization process:

The regulator synchronization resource has the following **important** fields:

Field	Required	Explanation
Name	Yes	This is the name of the facility
Regisration_number	Yes	This the registration number as assigned by the regulator
County	Yes	This is the code of the county where the facility is located
Owner	Yes	The id of the owner as the per the MFL
Facility_type	Yes	The id of the facility type as per the MFL
Mfl_code	No	The mfl code assigned to the facility once it is created in MFL

Obtaining the owner's ids

The owner's ids can be obtained by doing a `GET` to the **URL** `api/facilities/owners/`

Sample Expected Result:


```
{
    "count": 30,
    "next": null,
    "previous": null,
    "page_size": 30,
    "current_page": 1,
    "total_pages": 1,
    "start_index": 1,
    "end_index": 30,
    "results": [
        {
            "id": "aa1aca14-2937-4b49-b1a5-3c1ce05895ae",
            "owner_type_name": "Faith Based Organization",
            "created": "2015-09-23T13:16:13.988038Z",
            "updated": "2015-09-23T13:16:13.988060Z",
            "deleted": false,
            "active": true,
            "search": null,
            "name": "Faith Based",
            "description": null,
            "code": 1013,
            "abbreviation": null,
            "created_by": 1,
            "updated_by": 1,
            "owner_type": "c35677b6-05a4-4233-9dfa-9544476850c4"
        },
        {
```

```
            "id": "aa32ee6f-3653-4fb8-bd2e-4e59b61a952c",
            "owner_type_name": "Ministry of Health",
            "created": "2015-09-23T13:16:13.961025Z",
            "updated": "2015-09-23T13:16:13.961049Z",
            "deleted": false,
            "active": true,
            "search": null,
            "name": "Ministry of Health",
            "description": null,
            "code": 1010,
            "abbreviation": null,
            "created_by": 1,
            "updated_by": 1,
            "owner_type": "33ebff77-f5fc-46dd-b675-7abd56d7bfdf"
        }

    ]
}
```


Expected Response code: `HTTP_200_OK`

Obtaining the facility type ids

The facility type's ids can be obtained by doing a **GET** to the **URL** `api/facilities/facility_types/`
Sample Expected Result


```
{
    "count": 41,
    "next": "http://localhost:8061/api/facilities/facility_types/?page=2",
    "previous": null,
    "page_size": 30,
    "current_page": 1,
    "total_pages": 2,
    "start_index": 1,
    "end_index": 30,
    "results": [
        {
            "id": "1ce27507-9bd0-43cf-8a6f-4519a018ad27",
            "owner_type_name": null,
            "created": "2015-09-23T13:16:13.438542Z",
            "updated": "2015-09-23T13:16:13.438562Z",
            "deleted": false,
            "active": true,
            "search": null,
            "name": "Laboratory (Stand-alone)",
            "abbreviation": null,
            "sub_division": null,
            "created_by": 1,
            "updated_by": 1,
            "owner_type": null,
            "preceding": null
        },
        {
            "id": "f9f5bd67-b679-4711-8752-d77c2397ddc9",
            "owner_type_name": null,
            "created": "2015-09-23T13:16:13.431970Z",
            "updated": "2015-09-23T13:16:13.431993Z",
            "deleted": false,
            "active": true,
```

```
            "search": null,
            "name": "Hospital",
            "abbreviation": null,
            "sub_division": null,
            "created_by": 1,
            "updated_by": 1,
            "owner_type": null,
            "preceding": null
        }
    ]
}
```


Step 1

First the regulator system posts to MFL the details of the facilities that have been created in the RHIS and are not in the MFL. To do this do a `POST` to `api/facilitiess/regulator_sync/` a payload similar to the one below:


```
{
    "name": "Mama Lucy Kibaki hospital",
    "registration_number": 100,
    "county": 47,
    "owner": "aa1aca14-2937-4b49-b1a5-3c1ce05895ae",
    "facility_type": "f9f5bd67-b679-4711-8752-d77c2397ddc9"
}
```


Sample Expected Response:


```
{
    "id": "817c8a79-a3e5-46b1-aba5-4cb4de78a5da",
    "county_name": "NAIROBI",
    "owner_name": "Other Faith Based",
    "facility_type_name": "Hospital",
    "created": "2015-09-26T09:38:12.801942Z",
    "updated": "2015-09-26T09:38:12.801959Z",
    "deleted": false,
    "active": true,
    "search": null,
    "name": "Mama Lucy Kibaki hospital",
    "registration_number": "100",
    "county": 47,
    "mfl_code": null,
    "created_by": 4,
    "updated_by": 4,
    "facility_type": "f9f5bd67-b679-4711-8752-d77c2397ddc9",
    "owner": "aa1aca14-2937-4b49-b1a5-3c1ce05895ae"
}
```


Expected Response Code:

```
HTTP_201_CREATED
```

Step 2

Once a facility synchronization has been initiated, the request to register a facility will appear on the concerned CHRIO's dashboard. On registration of the facility with the MFL the mfl_code will be field and the RHRIS can now pull and get a facility's mfl_code.

Listing of synchronized facilities

To list the facilities requested do a `GET` to the URL `api/facilities/regulator_sync/`

Sample Expected Result:


```
{
    "count": 3,
    "next": null,
    "previous": null,
    "page_size": 30,
    "current_page": 1,
    "total_pages": 1,
    "start_index": 1,
    "end_index": 3,
    "results": [
        {
            "id": "817c8a79-a3e5-46b1-aba5-4cb4de78a5da",
            "county_name": "NAIROBI",
            "owner_name": "Other Faith Based",
            "facility_type_name": "Hospital",
            "created": "2015-09-26T09:38:12.801942Z",
            "updated": "2015-09-26T09:38:12.801959Z",
            "deleted": false,
            "active": true,
            "search": null,
            "name": "Mama Lucy Kibaki hospital",
            "registration_number": "100",
            "county": 47,
            "mfl_code": null,
            "created_by": 4,
            "updated_by": 4,
            "facility_type": "f9f5bd67-b679-4711-8752-d77c2397ddc9",
            "owner": "aa1aca14-2937-4b49-b1a5-3c1ce05895ae"
        },
        {
            "id": "94e91d84-6f73-48c1-855e-5a9539845971",
            "county_name": "GARISSA",
            "owner_name": "Private Practice - Medical Specialist",
            "facility_type_name": "Sub-District Hospital",
            "created": "2015-09-25T10:08:05.715148Z",
            "updated": "2015-09-25T10:08:05.715194Z",
            "deleted": false,
            "active": true,
            "search": null,
            "name": "Kamau Kiarie",
            "registration_number": "14535",
            "county": 7,
            "mfl_code": null,
            "created_by": 1,
            "updated_by": 1,
            "facility_type": "8b3b71b8-23ae-48a5-b7ee-e5078794c6c7",
            "owner": "a164b5bf-8caa-4558-9ba5-a77c87363b3d"
        },
        {
            "id": "f827f31d-aed0-4d63-90ad-7320769e4638",
            "county_name": "TAITA TAVETA",
            "owner_name": "Private Practice - Medical Specialist",
            "facility_type_name": "Sub-District Hospital",
```

```
            "created": "2015-09-25T10:07:54.192779Z",
            "updated": "2015-09-25T10:07:54.192817Z",
            "deleted": false,
            "active": true,
            "search": null,
            "name": "Mama Lucy",
            "registration_number": "14535",
            "county": 6,
            "mfl_code": null,
            "created_by": 1,
            "updated_by": 1,
            "facility_type": "8b3b71b8-23ae-48a5-b7ee-e5078794c6c7",
            "owner": "a164b5bf-8caa-4558-9ba5-a77c87363b3d"
        }
    ]
}
```


Part 2: Stand Alone Facilities Synchronization

Stand alone facilities are those facilities that offer only one specialized health care service. e.g laboratories, pharmacies, blood bank centers etc.

Synchronization process

Stand alone facilities such as pharmacies are registered in the regulator systems and are inspected and they start operating. On final inspection, the facilities are pushed to MFL via the API:

Pushing a Facility Basic Details

To push the details to MFL `POST` to `api/facilities/facilities/` a payload similar to the one below:


```
 {
    "owner": "af7f2be2-3454-4ba8-ae01-d24c05cfb382",
    "name": "Rehema Pharmacy (Bahati)",
    "official_name": "Rehema Pharmacy",
    "registration_number": "PBB 12444",
    "open_whole_day": true,
    "open_public_holidays": false,
    "open_weekends": true,
    "open_late_night": false,
    "plot_number": "LR/14414/KEN",
    "location_desc": "Along Chiefs Road",
    "facility_type": "6bbfc198-23f0-4310-9170-24ac05e2e49e",
    "operation_status": "3f5634c7-5a47-4e1d-b2f5-8e9b2308acf0",
    "ward": "b530b7ed-a110-431f-9a19-847eb706792d",
    "regulatory_body": "e4ae432e-8a0a-402c-ab6c-1c9033102bb5",
    "town": "ee724c13-abfe-44cb-98ce-9ec36a1e97a9"
}
```


The fields in the payload are explained below:

Field	Required	Explanation
name	Yes	This is the unique name of a facility e.g Agha Khan Medical Centre(Mombasa)
official_name	Yes	This is the name of the facility e.g Agha Khan medical centre
open_whole_day	No	Indicates whether a facility is open 24 hours a day
open_public_holidays	No	Indicates whether a facility is open on public holidays
open_late_night	No	Indicates whether a facility is open late night
open_weekends	No	Indicates whether a facility is open on weekends
plot_mumber	No	The plot number of where the facility is located
location_desc	No	A description on how to access the facility e.g which road to use
facility_type	Yes	This is the type of the facility <id> of the facility type e.f pharmacy
operation_status	Yes	The operation status id e.g Operation Status Id
ward	Yes	The ward ID of where the facility is located
regulatory_body	Yes	The regulatory body ID of the facility e.g Pharmacy and Poisons Board id
town	No	The id of the town or health centre where the facility is located
registration_number	Yes	This the registration number as assigned by the regulator
owner	Yes	The id of the owner as the per the MFL

Sample Expected Response:


```
{
    "id": "da3c4efe-57df-4d65-aa29-b6eb6719e469",
    "regulatory_status_name": "Pending License",
    "facility_type_name": "Pharmacy",
    "owner_name": "Private Practice - Unspecified",
    "owner_type_name": "Private Institutions and Private Practice",
    "owner_type": "2b8b031e-8d5a-47eb-b89c-a63d11e2b70a",
    "operation_status_name": "Operational",
    "county": "NAIROBI",
    "constituency": "MATHARE",
    "ward_name": "KIAMAIKO",
    "average_rating": 0,
    "facility_services": [],
    "is_approved": null,
    "has_edits": false,
    "latest_update": null,
    "regulatory_body_name": "Pharmacy & Poisons Board",
    "owner": "af7f2be2-3454-4ba8-ae01-d24c05cfb382",
    "date_requested": "2015-11-10T10:27:53.932Z",
    "date_approved": null,
    "latest_approval_or_rejection": null,
    "sub_county_name": null,
    "created": "2015-11-10T10:27:53.932878Z",
    "updated": "2015-11-10T10:27:53.932886Z",
    "deleted": false,
    "active": true,
    "search": null,
    "name": "Rehema Pharmacy (Bahati)",
    "official_name": "Rehema Pharmacy",
    "code": 100000,
    "registration_number": "PBB 12444",
    "abbreviation": null,
    "description": null,
    "number_of_beds": 0,
    "number_of_cots": 0,
    "open_whole_day": true,
```

```
    "open_public_holidays": false,
    "open_weekends": true,
    "open_late_night": false,
    "is_classified": false,
    "is_published": false,
    "attributes": null,
    "regulated": false,
    "approved": false,
    "rejected": false,
    "bank_name": null,
    "branch_name": null,
    "bank_account": null,
    "facility_catchment_population": null,
    "nearest_landmark": null,
    "plot_number": "LR/14414/KEN",
    "location_desc": "Along Chiefs Road",
    "closed": false,
    "closed_date": null,
    "closing_reason": null,
    "created_by": 4,
    "updated_by": 4,
    "facility_type": "6bbfc198-23f0-4310-9170-24ac05e2e49e",
    "operation_status": "3f5634c7-5a47-4e1d-b2f5-8e9b2308acf0",
    "ward": "b530b7ed-a110-431f-9a19-847eb706792d",
    "parent": null,
    "regulatory_body": "e4ae432e-8a0a-402c-ab6c-1c9033102bb5",
    "keph_level": null,
    "sub_county": null,
    "town": "ee724c13-abfe-44cb-98ce-9ec36a1e97a9",
    "contacts": []
}
```


Note: The MFL code assigned to the facility is in the response data (The field is `code`).

Expected Response Code:

```
HTTP_201_CREATED
```

It is clear that there is data that needs to be mapped between MFL and the regulators in order for this work.

The data includes:

1. Wards
2. Owners
3. Operation status
4. Facility types
5. Regulatory Bodies
6. Towns

All the above data is explained under the *Metadata resources* section of the documentation.

Pushing a Facility Geo-location Details

To push the geo-location details of a facility do a `POST` to the `URL` `api/gis/facility_coordinates/` with a payload similar to the one shown below:

```
{
    "source":"da488b76-2581-40d5-9377-3550e28cfb77", // The id of the source of the
↪geo-code
    "method":"4e1f460f-db3e-4e67-a906-2afc789f8f3a", // The id of the method used to
↪obtain the geo-code
    "collection_date":"2015-10-31T21:00:00.000Z", //Date when the geocode was obtained
    "facility":"da3c4efe-57df-4d65-aa29-b6eb6719e469", // The facility id to which
↪the geo-code belongs

    "coordinates": {
        "type":"Point",
        "coordinates":[

            36.87593521921288, // longitude
            -1.254507474965246 // latitude

        ]

    }

}
```

Expected Response Code `HTTP_201_CREATED`

Sample Expected Response

```
{
    "id":"c0d862d4-aa86-4cf0-9f10-8ec83b764321",
    "source_name":"DHMT Nakuru","method_name":"Taken with GPS Device",
    "created":"2015-11-10T10:45:01.731129Z",
    "updated":"2015-11-10T10:45:01.731141Z","deleted":false,
    "active":true,"search":null,

    "coordinates":{
        "type":"Point",
        "coordinates":[

            36.87593521921288,
            -1.254507474965246

        ]

    },

    "collection_date":"2015-10-31T21:00:00Z",
    "created_by":13,
    "updated_by":13,
    "facility":"da3c4efe-57df-4d65-aa29-b6eb6719e469",
    "source":"da488b76-2581-40d5-9377-3550e28cfb77",
    "method":"4e1f460f-db3e-4e67-a906-2afc789f8f3a"

}
```

After obtaining the id from the response data do a PATCH to the URL api/facilities/facilities/<facility_id> with a payload similar to the one shown below:

```
{
    "coordinates": "c0d862d4-aa86-4cf0-9f10-8ec83b764321" // the id obtained from the␣
↪response data above

}
```

Expected Response Code: HTTP_204_NO_CONTENT

There is no response data.

Note:

1. The **geo-code methods** ids are obtained from the endpoint api/gis/geo_code_methods/
2. The **geo-code sources** ids are obtained from the endpoint api/gis/geo_code_sources/
3. Watch out for the order of the coordinates; the longitude comes before the latitude otherwise the geocodes will not validate.

Pushing a Facility Contacts

To push a facility's contacts do a PATCH to the URL api/facilities/facilities/<facility_id> with a payload similar to the one shown below:

```
{
    "contacts":[
        {
            "contact_type":"17287e65-021f-4319-92fb-e032e2c3de72", // the contact␣
↪type id
            "contact":"0200046" // the actual contact

        },

        {
            "contact_type":"9417c555-e36f-4502-941e-9a9943c534d5",
            "contact":"1414141241"

        }

    ]

}
```

Expected Response code: HTTP_204_NO_CONTENT

There is no response data

Note: The **contact types** ids can be obtained from the endpoint api/common/contact_types/

Pushing a Facility's Officer-In-Charge

To push the details of a facility's officer-in-charge to a `PATCH` to the `URL` `api/facilities/facilities/<facility_id>` with a payload similar to the one shown below:

```
{
    "officer_in_charge":{
        "name":"Alex Aluoch",
        "reg_no":"P15/3525/5235",

        "contacts":[
            {
                "type":"17287e65-021f-4319-92fb-e032e2c3de72", // the contact type id
                "contact":"020133555" // the actual contact

            },

            {
                "type":"d7c0405c-1f69-4d1d-9895-24e6af997429",
                "contact":"0756456288"

            }

        ],

        "title":"ba36158a-0d61-4014-aa55-111425b06775" // the job title id

    }

}
```

Expected Response code: `HTTP_204_NO_CONTENT`

There is no response data

Note: To obtain the **job-titles** go to the `URL` `api/facilities/job_titles/`

Pushing a Facility's Services

To push a facility's services do a `PATCH` to the `URL` `api/facilities/facilities/<facility_id>` with a payload similar to the one shown below:

```
{
    "services":[
        {
            "service":"59c4e20e-eb00-427c-8533-61719b0db77d" // the service id

        },

        {
            "service":"78aac8b6-c2d7-4204-b074-8b83fb1ef070", // the service id
            "option":"888c5b48-2334-436d-a806-3a57e1933e8b" // the option id

        },
```

```
        {
            "service":"576c9964-ee5a-4a6f-b1fd-32064d76bb77" // the service id

        }

    ]

}
```

Expected Response code: `HTTP_204_NO_CONTENT`

There is no response data

Note: Services in MFL

All the service in MFL can be obtained from the `URL` `api/facilities/services/`

It is important to note that there are two types of services in the MFL:

a). Services with options

b). Services without options

For the services that do not have options only the service id is posted and when the service has an option such as basic or comprehensive the service id is posted together with the option id as the payload above shows.

Each service from the endpoint `api/facilities/services/` comes together with its **group** and from the group object the **options** can be obtained.

To list all the option groups do a `GET` to `api/facilities/option_groups/` and to get the details of one single option group do a `GET` to `api/facilities/option_groups/<option_group_id>`

Once details of a facility have been pushed to MFL, all the facility details can be obtained through doing a `GET` to `api/facilities/facilities/<facility_id>`

For example a `GET` to `api/facilities/facilities/da3c4efe-57df-4d65-aa29-b6eb6719e469/` would result in the details of the facility that was created and updated in the sample payloads above.


```
{
    "id": "da3c4efe-57df-4d65-aa29-b6eb6719e469",
    "regulatory_status_name": "Pending License",
    "facility_type_name": "Pharmacy",
    "owner_name": "Private Practice - Unspecified",
    "owner_type_name": "Private Institutions and Private Practice",
    "owner_type": "2b8b031e-8d5a-47eb-b89c-a63d11e2b70a",
    "operation_status_name": "Operational",
    "county": "NAIROBI",
    "constituency": "MATHARE",
    "ward_name": "KIAMAIKO",
    "average_rating": 0.0,

    "facility_services": [
        {
            "average_rating": 0.0,
            "category_id": "edd7631d-b2f3-4008-9c76-e3abb68a547d",
            "number_of_ratings": 0,
            "option": null,
            "service_name": "Short Term FP",
            "option_name": "Yes",
```

```
            "service_id": "576c9964-ee5a-4a6f-b1fd-32064d76bb77",
            "service_code": 1012,
            "id": "f053976f-3b0a-42ce-8b92-4695cce1bbf0",
            "category_name": "Family Planning"

        },

        {
            "average_rating": 0.0,
            "category_id": "edd7631d-b2f3-4008-9c76-e3abb68a547d",
            "number_of_ratings": 0,
            "option": "888c5b48-2334-436d-a806-3a57e1933e8b",
            "service_name": "Permanent FP",
            "option_name": "Level 3",
            "service_id": "78aac8b6-c2d7-4204-b074-8b83fb1ef070",
            "service_code": 1051,
            "id": "bbb2ce77-5537-43c5-9364-66c307b21c6a",
            "category_name": "Family Planning"

        },

        {
            "average_rating": 0.0,
            "category_id": "edd7631d-b2f3-4008-9c76-e3abb68a547d",
            "number_of_ratings": 0,
            "option": null,
            "service_name": "Long Term FP",
            "option_name": "Yes",
            "service_id": "59c4e20e-eb00-427c-8533-61719b0db77d",
            "service_code": 1013,
            "id": "985d94c3-409b-4a5a-b53d-f9589d338d68",
            "category_name": "Family Planning"

        }

    ],

    "is_approved": null,
    "has_edits": false,
    "latest_update": null,
    "regulatory_body_name": "Pharmacy & Poisons Board",
    "owner": "af7f2be2-3454-4ba8-ae01-d24c05cfb382",
    "date_requested": "2015-11-10T10:27:53.932Z",
    "date_approved": null,
    "latest_approval_or_rejection": null,
    "sub_county_name": null,

    "facility_contacts": [
        {
            "contact_type_name": "FAX",
            "contact": "1414141241",
            "id": "341c5ea6-9d85-47f2-b635-12ee013c7da7",
            "contact_id": "e729ea90-2dde-4361-95f2-fd94c059f56b"

        },

        {
            "contact_type_name": "LANDLINE",
```

```
            "contact": "0200046",
            "id": "e0a85e74-47fa-4112-aa76-fd9dccd2fb89",
            "contact_id": "95395096-0a61-4b5b-88d8-123402eb86ba"

        }

    ],
    "coordinates": "c0d862d4-aa86-4cf0-9f10-8ec83b764321",
    "latest_approval": null,

    "boundaries": {
        "county_boundary": "d89fad95-0f7d-4044-87ec-f8a7ad9fcac2",
        "ward_boundary": "0f8d1126-d978-4f3e-afe8-e46635ddc0fe",
        "constituency_boundary": "0f0ecac3-fb36-450d-bd93-efb8558a1a1e"

    },

    "service_catalogue_active": true,
    "facility_units": [],

    "officer_in_charge": {
        "name": "Alex Aluoch",

        "contacts": [
            {
                "contact_type_name": "MOBILE",
                "officer_contact_id": "fe69052a-4466-45ce-ab64-80d7f7c1eef8",
                "type": "d7c0405c-1f69-4d1d-9895-24e6af997429",
                "contact_id": "edb51ac8-71e2-477a-852b-f90ff3152973",
                "contact": "0756456288"

            },

            {
                "contact_type_name": "LANDLINE",
                "officer_contact_id": "ae3cbec5-f9e5-4ec0-9465-af5f13773256",
                "type": "17287e65-021f-4319-92fb-e032e2c3de72",
                "contact_id": "d8f51a01-a0a5-49d8-9be9-0ae0a4c604cc",
                "contact": "020133555"

            }

        ],

        "id_number": null,
        "reg_no": "P15/3525/5235",
        "title": "ba36158a-0d61-4014-aa55-111425b06775",
        "title_name": "Medical Superintendant"

    },

    "town_name": "Bahati",
    "keph_level_name": null,
    "created": "2015-11-10T10:27:53.932878Z",
    "updated": "2015-11-10T10:27:53.932886Z",
    "deleted": false,
    "active": true,
    "search": null,
```

```
    "name": "Rehema Pharmacy (Bahati)",
    "official_name": "Rehema Pharmacy",
    "code": 100000,
    "registration_number": "PBB 12444",
    "abbreviation": null,
    "description": null,
    "number_of_beds": 0,
    "number_of_cots": 0,
    "open_whole_day": true,
    "open_public_holidays": false,
    "open_weekends": true,
    "open_late_night": false,
    "is_classified": false,
    "is_published": false,
    "regulated": false,
    "approved": false,
    "rejected": false,
    "bank_name": null,
    "branch_name": null,
    "bank_account": null,
    "facility_catchment_population": null,
    "nearest_landmark": null,
    "plot_number": "LR/14414/KEN",
    "location_desc": "Along Chiefs Road",
    "closed": false,
    "closed_date": null,
    "closing_reason": null,
    "created_by": 4,
    "updated_by": 4,
    "facility_type": "6bbfc198-23f0-4310-9170-24ac05e2e49e",
    "operation_status": "3f5634c7-5a47-4e1d-b2f5-8e9b2308acf0",
    "ward": "b530b7ed-a110-431f-9a19-847eb706792d",
    "parent": null,
    "regulatory_body": "e4ae432e-8a0a-402c-ab6c-1c9033102bb5",
    "keph_level": null,
    "sub_county": null,
    "town": "ee724c13-abfe-44cb-98ce-9ec36a1e97a9",

    "contacts": [
        "e729ea90-2dde-4361-95f2-fd94c059f56b",
        "95395096-0a61-4b5b-88d8-123402eb86ba"

    ]

}
```