DinoMail

Release 1.1.0

Dec 26, 2021

Contents:

1 Installation **3**
- 1.1 Database . . . 3
- 1.2 Clone and install dependencies . . . 3
- 1.3 Settings . . . 3
- 1.4 Run migration, create a superuser and run the app . . . 7
- 1.5 Statics . . . 7
- 1.6 Translation . . . 7

2 Quickstart **9**
- 2.1 Introduction . . . 9
- 2.2 Virtual Domains . . . 9
- 2.3 Virtual Users . . . 11
- 2.4 Virtual Aliases . . . 12

3 Linking DinoMail to postfix and dovecot **13**
- 3.1 Postfix . . . 13
- 3.2 Dovecot . . . 15

4 API **17**
- 4.1 Authentication . . . 17
- 4.2 Virtual domains, users and aliases urls . . . 17

Index **19**

DinoMail is a web interface that helps you managing your mail server.

This is documention for DinoMail version 1.1.0.

CHAPTER 1

Installation

1.1 Database

You will need a working database for this project. We recommend the use of PostgreSQL but any database working with django (see here) will do the trick.

However, please note that this software should be linked to mail server softwares like postfix and dovecot and that any datbase might not work with those. Please see the documentations of those before choosing any database.

You will need to create, before installatiom, a database (e.g. `dinomail`) and a user (e.g. `dinomail`) with some password that we will denote `secret` for the rest of this page.

1.2 Clone and install dependencies

First you will have to clone the Github repository of the project. We recommmend you to clone from the last release.

```
git clone https://github.com/nanoy42/dinomail
```

Then you need to install the dependencies. There is a Pipfile, from which you can just do

```
pipenv install
```

Or you can use the requirements.txt file :

```
pip3 install -r requirements.txt
```

1.3 Settings

In the `src/dinomail` folder there is file `local_settings.example.py`. Copy it in the same directory as `local_settings.py` :

```
cp src/dinomail/local_settings.example.py src/dinomail/local_settings.example.py
```

Next, you will have to edit this file to change the settings. Some are django settings, some are api settings and some are DinoMail specific settings. They are described below.

1.3.1 Django settings

Please see the django documentation for extended documentation.

SECRET_KEY

A secret key for a particular Django installation. This is used to provide cryptographic signing, and should be set to a unique, unpredictable value. This value should be kept secret.

DEBUG

A boolean that turns on/off debug mode. You should use `DEBUG=False` for production.

ALLOWED_HOSTS

A list of strings representing the host/domain names that this Django site can serve.

DATABASES

If you use a postgresql database, on the same host as where you installed DinoMail, with the above values, it should look like this:

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql",
        "NAME": "dinomail",
        "USER": "dinomail",
        "PASSWORD": "secret",
        "HOST": "localhost",
    }
}
```

LANGUAGE_CODE

User will not be able to change the interface language. However, you can select the language you want from the listed below :

- English (`'en'`)
- French (`'fr'`)

Those are the languages currently supported for DinoMail.

TIME_ZONE

The time zone to use.

STATIC_ROOT

Folder in which the static files should be copied.

1.3.2 API settings

Tastypie is used for the API. One setting is set in the `local_settings.example.py` :

API_LIMIT_PER_PAGE

Default number of object to display when an api request is made. 0 stands for no limit. Default (in DinoMail) is 0.

Note: If the value is not set, the default value from tastypie is 20.

There are some other settings from tastypie, you can see them here.

1.3.3 DinoMail settings

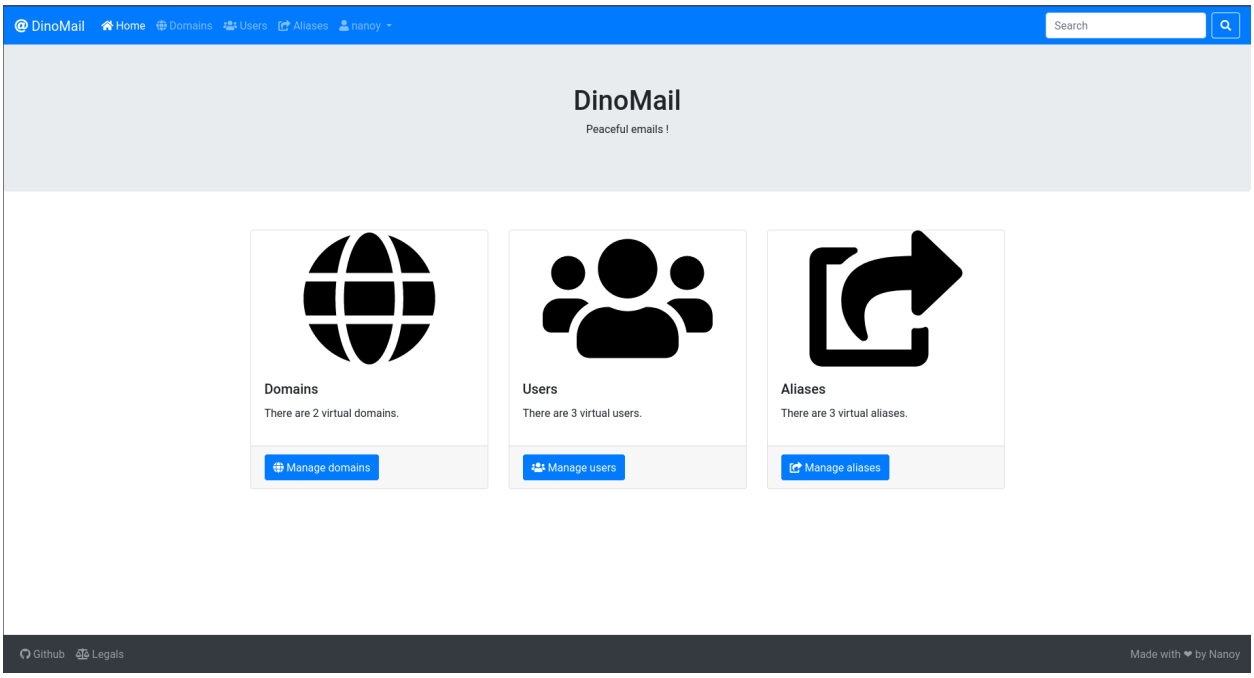
There are three DinoMail specific settings:

DINOMAIL_NAME

The name that will appear in the navbar, in the tab, on the login page and on some other places on the interface. Default is `DinoMail`.

DINOMAIL_CATCH_LINE

Sentence displayed on the home page.



DINOMAIL_LEGALS

Paragraph displayed on the legals page.

DINOMAIL_PASSWORD_SCHEME

Dovecot supports various password schemes. DinoMail supports most of them. By default the salted-sha512 is used.

Note: By default, dovecot use the bcrypt scheme, but this scheme requires an extra python library.

Here are the supported schemes:

- PLAIN (`core.utils.make_password_plain`)

- PLAIN-TRUNC (`core.utils.make_password_plain_trunc`)
- CLEARTEXT (`core.utils.make_password_cleartext`)
- CLEAR (`core.utils.make_password_clear`)
- SHA (`core,utils.make_password_sha`)
- SSHA (`core.utils.make_password_ssha`)
- SHA256 (`core.utils.make_password_sha256`)
- SSHA256 (`core.utils.make_password_ssha256`)
- SHA512 (`core.utils.make_password_sha512`)
- SSHA512 (`core.utils.make_password_ssha512`)
- PLAIN-MD5 (`core.utils.make_password_plain_md5`)
- LDAP-MD5 (`core.utils.make_password_ldap_md5`)
- CRYPT (`core.utils.make_password_crypt`)
- DES-CRYPT (`core.utils.make_password_des_crypt`)
- MD5-CRYPT (`core.utils.make_password_md5_crypt`)
- SHA256-CRYPT (`core.utils.make_password_sha256_crypt`)
- SHA512-CRYPT (`core.utils.make_password_sha512_crypt`)

This schemes are also supported, installing an extra library:

- ARGON2I (`core.utils_argon.make_password_argon2i`) and ARGON2ID (`core.utils_argon.make_password_argon2id`) installing `argon2-cffi`.
- BLF-CRYPT (`core.utils_bcrypt.make_password_blf_crypt`) installing `bcrypt`.
- LANMAN (`core,utils_passlib.make_password_lanman`) installing `passlib`.

Warning: Some of these schemes are considered *unsecure*. Even if there are supported, please don't use them. Use salted hashing algorithms.

The following algorithms are supported by Dovecot but not by DinoMail:

- HMAC-MD5
- OTP
- RPA
- SKEY
- PLAIN-MD4
- SCRAM-SHA-1
- NTLM
- MD5
- PBKDF2
- CRAM-MD5
- SMD5

- DIGEST-MD5

1.4 Run migration, create a superuser and run the app

To run migrations (i.e. create the database schema), you need to run the following command after setting the database :

```
python3 manage.py migrate
```

You can then create a superuser with the command

```
python3 manage.py createsuperuser
```

You will be prompted for some information.

You can check the intstallation by running :

```
python3 manage.py check
```

Then you can test the app with

```
python3 manage.py runserver 0.0.0.0:8000
```

Warning: You should not use runserver for production. Instead, use wsgi modules for apache or nginx by instance.

1.5 Statics

The good option is to serve the statics directly with your web server. Then if `DEBUG` is set to False, DinoMail will not serve the statics.

You have to set the `STATIC_ROOT` settings and execute the

```
python3 manage.py collectstatic
```

and configure your web server. For example, for apache, you could add the following line :

```
Alias /static/ /var/www/dinomail/static
```

if you have set `STATIC_ROOT` to `/var/www/dinomail/static` for example.

1.6 Translation

To compile translation files, use the following command :

```
django-admin compilemessages
```

CHAPTER 2

Quickstart

2.1 Introduction

This page quickly explains the functionnalities of DinoMail.

Almost any operation can be made using the admin site of django, in fact all except 2 which are the modification of a user's password and the regeneration of an api key.

You can access the admin and regenerate your apikey by clicking on your username in the navbar and choosing the appropriate option.

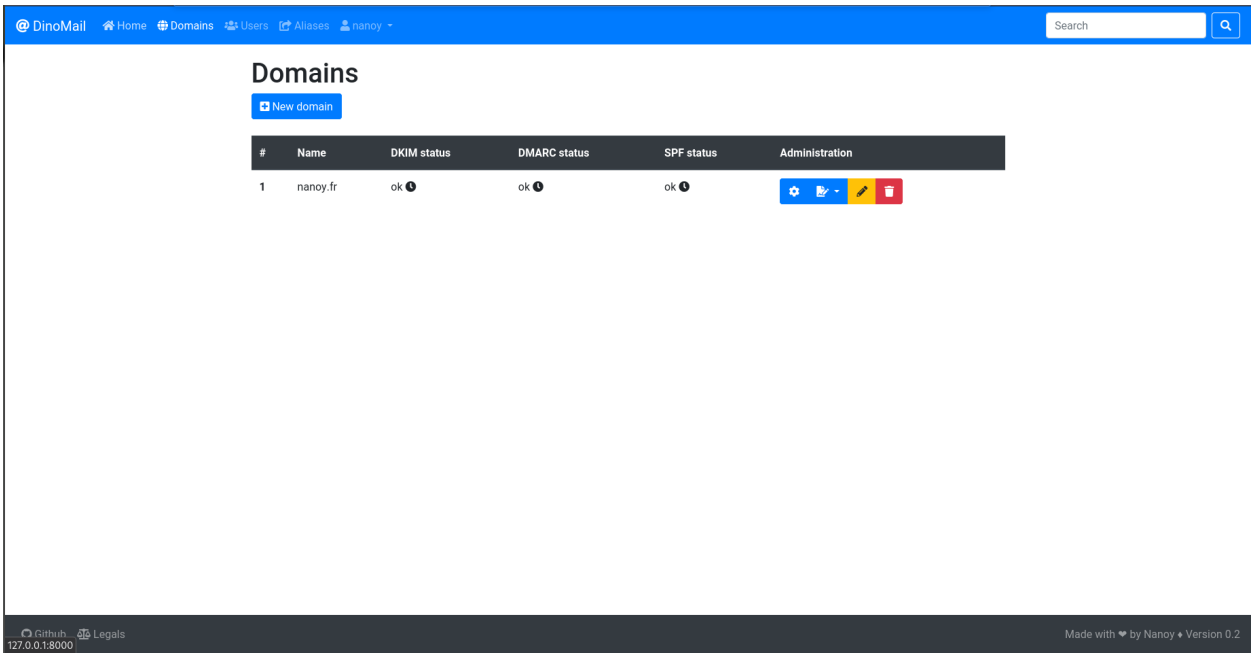
2.2 Virtual Domains

2.2.1 Introduction

Virtual domains are domains for which your mail server is responsible for.

2.2.2 Manage virtual domains

You can create, change and delete virtual domains on the virtual domains page :



On each row :

- The first button (the cog) downloads the xml autoconfig file.
- The second button (the signature) gives two choices : update the DKIM status or see the DKIM scan details.
- The third button (the pencil) displays a form to change the domain.
- The last button (the bin) deletes the domain (you will be asked to confirm deletion).

2.2.3 Fields

`VirtualDomain.`**`name`**

Required

Name of the domain, in the form `example.org`.

This attribute should be unique.

`VirtualDomain.`**`dkim_key_name`**

Dkim key name (the one in the DNS record name).

`VirtualDomain.`**`dkim_key`**

Value of the dkim public key (the one stored in the DNS TXT record).

`VirtualDomain.`**`display_name`**

Display name for the xml autoconfig file.

`VirtualDomain.`**`short_display_name`**

Short display name for the xml autoconfig file.

`VirtualDomain.`**`imap_address`**

The imap address for the xml autoconfig file. If not set, `imap.(name of domain)` will be used.

`VirtualDomain.`**`pop_address`**

The pop address for the xml autoconfig file. If not set, the pop section will be skipped.

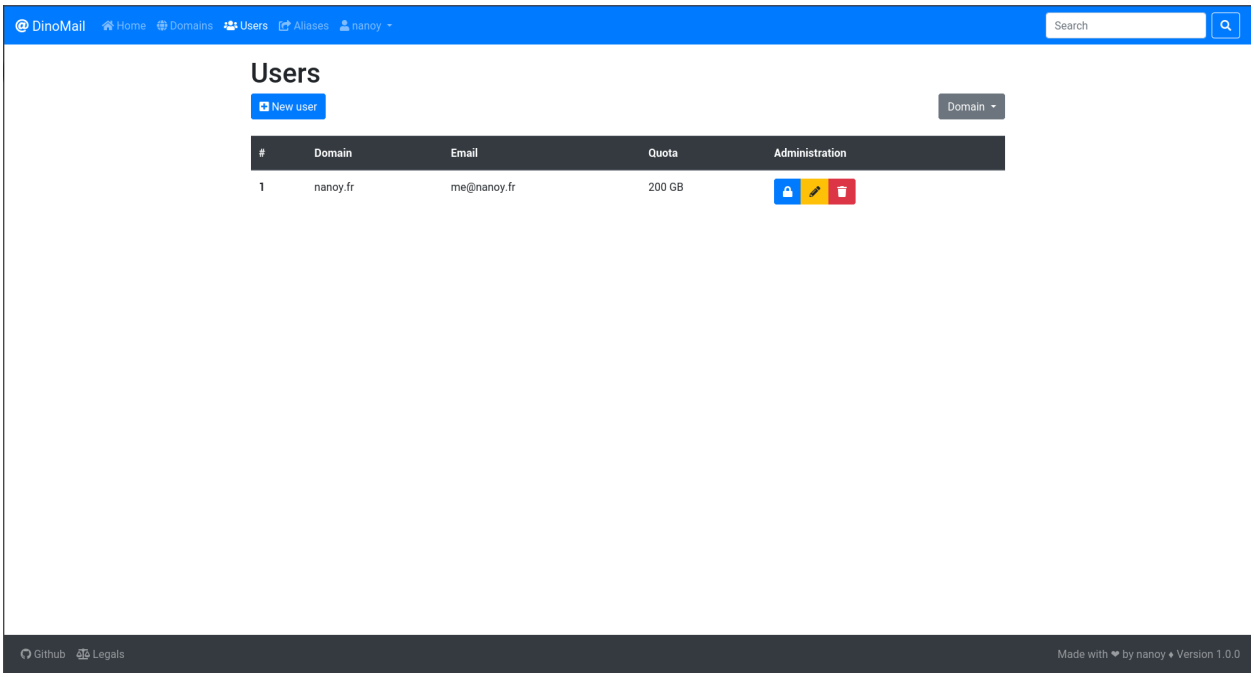
`VirtualDomain.`**`smtp_address`**

The smtp address for the xml autoconfig file. If not set, `smtp.(name of domain)` will be used.

2.3 Virtual Users

2.3.1 Manage virtual users

You can create, change. change password and delete virtual users which are actual mailboxes.



On each row :

- The first button (the lock) displays a form to change user's password.
- The second button (the pencil) displays a form to change the user.
- The last button (the bin) deletes the user (you will be asked to confirm deletion).

> **Warning:** When creaing or modifying a user, the quota should be given in **bytes**.

2.3.2 Fields

`VirtualUser.`**`domain`**

Required

The VirtualDomain of the account.

`VirtualUser.`**`email`**

Required

The email of the account.

This attribute should be unique.

`VirtualUser.`**`password`**

Required

The hashed password of the account (in a dovecot-compliant way).

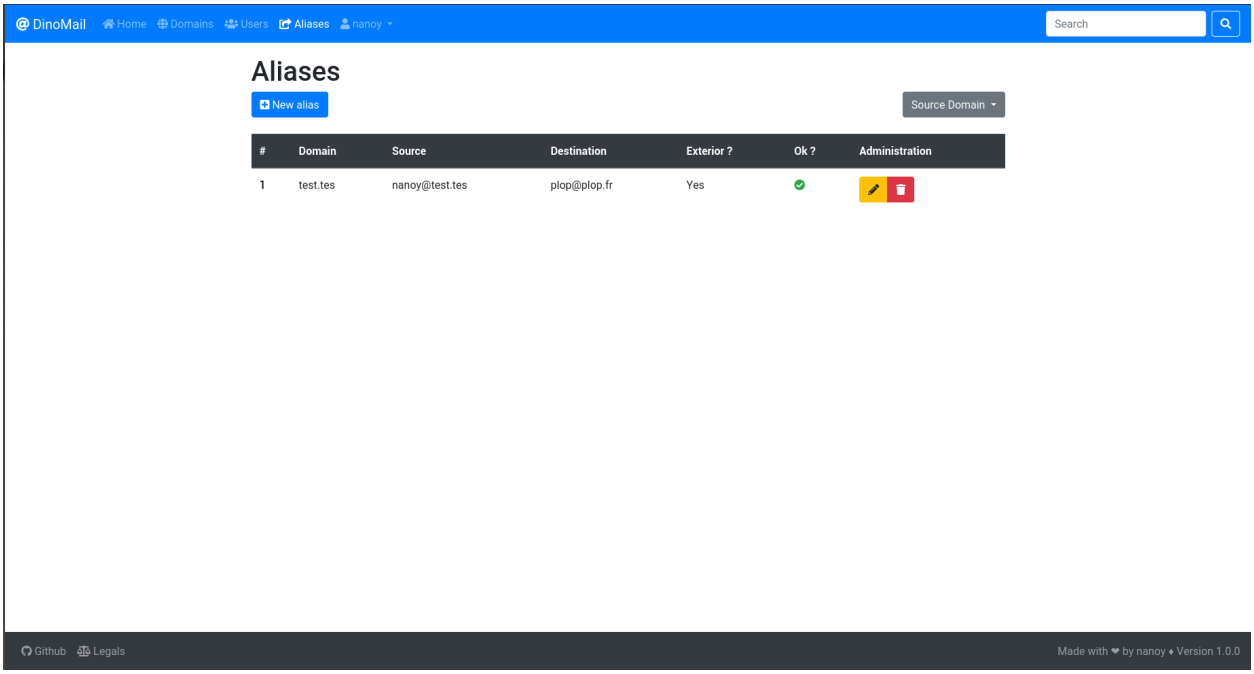
`VirtualUser.`**`quota`**

Required

The quota of the account, in **bytes**.

2.4 Virtual Aliases

You can create, change and delete virtual aliases which are fake email account that redirects to true ones.



On each row :

- Exterior means if the domain of the **destination** email is handled by the current instance or not.
- Ok ? verifies, in the case of an interior email, if the **destination** email exists as an alias or a virtual user in the database.
- The first button (the lock) displays a form to change user's password.
- The second button (the pencil) displays a form to change the user.
- The last button (the bin) deletes the user (you will be asked to confirm deletion).

CHAPTER 3

Linking DinoMail to postfix and dovecot

Now we are going to see how to link DinoMail data to postfix and dovecot in order to have a working email server.

Note: This is an example implementation. Other implementations could work. This is mainly inspired by https://workaround.org/ispmail.

We will use a postgresql database, but it could work with other types of database. We redirect to the documentations of postfix and dovecot. The user is `dinomail` with password `secret`. The database is `dinomail`.

Warning: Additional packages are needed for databases with postfix and dovecot (postfix-pgsql and dovecot-pgsql or postfix-mysql and dovecot-mysql for debian packages).

3.1 Postfix

3.1.1 Create configuration files

First we create a pgsql.d directory in `/etc/postfix` :

```
mkdir /etc/postfix/pgsql.d
```

Next we are going to populate the virtual domains :

```
# /etc/postfix/pgsql.d/virtual-mailbox-domains.cf
user = dinomail
password = secret
hosts = 127.0.0.1
dbname = dinomail
query = SELECT 1 FROM core_virtualdomain WHERE name='%s'
```

Note: We select 1 because we must have a map with the key being the domain and the value is meaningless.

We do the same for virtual users and virtual aliases :

```
# /etc/postfix/pgsql.d/virtual-mailbox-maps.cf
user = dinomail
password = secret
hosts = 127.0.0.1
dbname = dinomail
query = SELECT 1 FROM core_virtualuser WHERE email='%s'
```

and

```
# /etc/postfix/pgsql.d/virtual-alias-maps.cf
user = dinomail
password = secret
hosts = 127.0.0.1
dbname = dinomail
query = SELECT destination FROM core_virtualalias WHERE source='%s'
```

We can also add an email2email.cf file which is a map where the key is an email and the value is the same email. It can be useful for wildcard matches for example :

```
# /etc/postfix/pgsql.d/email2email.cf
user = dinomail
password = secret
hosts = 127.0.0.1
dbname = dinomail
query = SELECT email FROM core_virtualuser WHERE email='%s'
```

3.1.2 Test the configuration files

You can use the `postmap` utility. If you have the domain `example.org`, a user `test@example.org` and an alias `postmaster@example.org -> test@example.org`.

```
postmap -q example.org pgsql:/etc/postfix/pgsql.d/virtual-mailbox-domains.cf
postmap -q test@example.org pgsql:/etc/postfix/pgsql.d/virtual-mailbox-maps.cf
postmap -q postmaster@example.org pgsql:/etc/postfix/pgsql.d/virtual-alias-maps.cf
postmap -q test@example.org pgsql:/etc/postfix/pgsql.d/email2email.cf
```

It should return 1,1, test@example.org and test@example.org.

3.1.3 Make postfix use the configuration files

Next, we have to specify postfix how these maps should be used. You can edit the `main.cf` file in `/etc/postfix` or use the `postconf` utility :

```
postconf virtual_mailbox_domains=pgsql:/etc/postfix/pgsql.d/virtual-mailbox-domains.cf
postconf virtual_mailbox_maps=pgsql:/etc/postfix/pgsql.d/virtual-mailbox-maps.cf
postconf virtual_alias_maps=pgsql:/etc/postfix/pgsql.d/virtual-alias-maps.cf
```

3.2 Dovecot

Next we want Dovecot to use the information in the database too. We edit `/etc/dovecot/conf.d/auth-sql.conf.ext` and set

```
userdb {
    driver = sql
    args = /etc/dovecot/dovecot-sql.conf.ext
}
```

Then we edit `/etc/dovecot/dovecot-sql.conf.ext` and set

```
driver = pgsql
connect = host=127.0.0.1 dbname=dinomail user=dinomail password=secret
user_query = SELECT email as user, \
concat('*:bytes=', quota) AS quota_rule, \
'/var/vmail/%d/%n' AS home, \
5000 AS uid, 5000 AS gid \
FROM core_virtualuser WHERE email='%u'
password_query = SELECT password FROM core_virtualuser WHERE email='%u'
iterate_query = SELECT email AS user FROM core_virtualuser
```

CHAPTER 4

API

DinoMail exposes an API. The API allows to :

- get a token
- list, detail, create, modify and delete a virtual domain
- list, detail, create, modify, modify password of and delete a virtual user
- list, detail, create, modify and delete a virtual alias

4.1 Authentication

For the most part of the API, you will need to authenticate with an ApiKey : `Authorization: ApiKey username:apikey`.

To get an api key, you can query the url `/api/apikey/` with a basic authentication : `Authorization: Basic username:password`.

Note: If you're using Apache & mod_wsgi, you will need to enable WSGIPassAuthorization On.

4.2 Virtual domains, users and aliases urls

The basis URLS are

- `/api/virtualdomain`
- `/api/virtualuser`
- `/api/virtualalias`

You can then, for each of those basis urls, query

- / (GET) : list of all objects.
- / (POST) : add an object.
- /<pk>/ (GET) : get detail for object with primary key <pk>.
- /<pk>/ (PUT) : update all fields of object with primary key <pk>.
- /<pk>/ (PATCH) : update specified fields of object with primayr key <pk>.
- /<pk>/ (DELETE) : delete object with primary key <pk>.

Note: A POST on /api/virtualuser/<pk>/ will not reset the password.

There is also a special URL to change a user's password : /api/changeuserpassword/<pk>/, (POST or PATCH are available). You have to transmit the plain text password.

You can take a look at https://django-tastypie.readthedocs.io/en/latest/interacting.html/.

Index

A

ALLOWED_HOSTS, 4
API_LIMIT_PER_PAGE, 4

D

DATABASES, 4
DEBUG, 4
DINOMAIL_CATCH_LINE, 5
DINOMAIL_LEGALS, 5
DINOMAIL_NAME, 5
DINOMAIL_PASSWORD_SCHEME, 5
display_name (*VirtualDomain attribute*), 10
dkim_key (*VirtualDomain attribute*), 10
dkim_key_name (*VirtualDomain attribute*), 10
domain (*VirtualUser attribute*), 11

E

email (*VirtualUser attribute*), 11

I

imap_address (*VirtualDomain attribute*), 10

L

LANGUAGE_CODE, 4

N

name (*VirtualDomain attribute*), 10

P

password (*VirtualUser attribute*), 12
pop_address (*VirtualDomain attribute*), 10

Q

quota (*VirtualUser attribute*), 12

S

SECRET_KEY, 4
short_display_name (*VirtualDomain attribute*), 10
smtp_address (*VirtualDomain attribute*), 11
STATIC_ROOT, 4

T

TIME_ZONE, 4