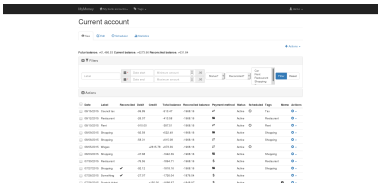
MyMoney Documentation

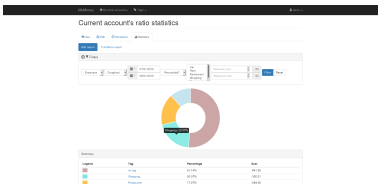
Release 1.0

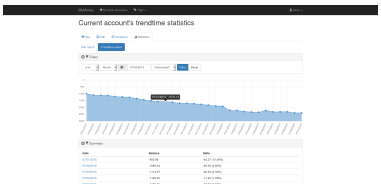
Yannick Chabbert

October 24, 2015

Contents







This is the documentation of the MyMoney project, a personal finance Web application build with the Django framework.

Of course, just like this project, this documentation isn't really useful. This is mainly a good use case to learn Sphinx and ReadTheDoc :)

CHAPTER 1

Contents

1.1 Installation

1.1.1 Requirements

- Python 3.4 (no backward compatibility)
- PostgreSQL **only** (no MySQL or SQLite support)

1.1.2 Deployment

Backend

The deployment is the same as any other Django projects. Here is a quick summary:

1. install required system packages. For example on Debian:

```
apt-get install python3 python3-dev postgresql-9.4 libpq-dev virtualenv
```

2. create a PostgreSQL database in a cluster with role and owner
3. create a virtualenv:

```
virtualenv <NAME> -p python3
```

4. install dependencies with pip (see *Production* or *Development*)
5. configure the settings (see *Production* or *Development*)
6. export the `DJANGO_SETTINGS_MODULE` to easily use the `manage.py` with the proper production setting. For example:

```
export DJANGO_SETTINGS_MODULE="mymoney.settings.production"
```

7. import the SQL schema:

```
./manage.py migrate
```

8. create a super user:

```
./manage.py createsuperuser
```

Note: WSGI will use the `production.py` settings, whereas `manage.py` will use the `local.py` by default.

Production

- Install dependencies (in virtualenv):

```
pip install -r requirements/production.txt
```

- copy `mymoney/settings/production.dist` to `mymoney/settings/production.py` and edit it:

```
cp mymoney/settings/production.dist mymoney/settings/production.py
```

- install JS libraries **first** with *Bower* (see *Frontend*) then collect statics files:

```
./manage.py collectstatic
```

- execute the Django check command and apply fixes if needed:

```
./manage.py check --deploy
```

- Set up cron tasks on server to execute the following commands:
  - cloning recurring bank transactions:

```
./manage.py clonescheduled
```

  - cleanup tasks (only usefull with further user accounts):

```
./manage.py deleteorphansbankaccounts
```

At the project root directory, the `scripts` directory provides bash script wrappers to execute these commands. Thus, you could create cron rules similar to something like:

```
0 1 * * *  ABSOLUTE_PATH/scripts/clonescheduled.sh <ABSOLUTE_PATH_TO_V_ENV>
0 2 * * *  ABSOLUTE_PATH/scripts/deleteorphansbankaccounts.sh <ABSOLUTE_PATH_TO_V_ENV>
```

For example, create a file in `/etc/cron.d/clonescheduled`, and edit:

```
0 2 * * * <USER> /ABSOLUTE_PATH/scripts/clonescheduled.sh <ABSOLUTE_PATH_TO_V_ENV>
```

Development

- Install dependencies:

```
pip install -r requirements/local.txt
```

- copy `mymoney/settings/local.dist` to `mymoney/settings/local.py` and edit it:

```
cp mymoney/settings/local.dist mymoney/settings/local.py
```

Frontend

1. install Bower. One way is to do it with npm globally:

```
npm install -g bower
```

2. At the project root directory, run the following command to install JS libraries dependencies:

```
bower install --production
```

Development

1. install *gulp* globally to use it as a command line tool:

```
npm install -g gulp
```

2. go to the project root directory and install gulp dependencies:

```
npm install
```

3. once *node* packages are installed *locally* in `./node_modules`, you should be able to execute the following gulp commands implemented in `gulpfile.js`:

   - *js*: concat and minify js
   - *css*: concat and minify css

   To execute all commands at once, from the project root directory, just execute:

```
gulp
```

1.1.3 Internationalization

1. copy `mymoney/settings/l10n.dist` to `mymoney/settings/l10n.py` and edit it:

```
cp mymoney/settings/l10n.dist mymoney/settings/l10n.py
```

   Further notes about some additional settings:

   - `USE_L10N_DIST`: Whether to use the minify file including translations. It imply that the translated file is generated with *gulp* (`mymoney.min.<LANGCODE>.js`). If false (default), additionnal JS translations files would be loaded.
   - `BOOTSTRAP_CALENDAR_LANGCODE`: If `USE_L10N_DIST` is false, the language code to use to load the translation file at: `mymoney/static/bower_components/bootstrap-calendar/js/language/<LANG`
   - `BOOTSTRAP_DATEPICKER_LANGCODE`: If `USE_L10N_DIST` is false, the language code to use to load the translation file at: `mymoney/static/bower_components/bootstrap-datepicker/js/locales/bootstrap-datepicke`

2. edit your final setting file to use the l10n configuration instead:

```
# from .base import *
from .l10n import *
```

3. optionally build the minified JS distribution for your language. To achieve it, you first need to have *gulp* installed. See section *Development* for more details about *gulp*. The `gulp js` accept optional parameters:

   - `--lang`: the IETF language code of the form : *xx-XX*. **Must** be the same as the Django `LANGUAGE_CODE` setting.
   - `--lang_bt_cal`: the Bootstrap calendar language code to use. To see the list of available code supported, take a look at : `mymoney/static/bower_components/bootstrap-calendar/js/language/<LANGCODE>.js`

- `--lang_bt_dp`: the Bootstrap datepicker language code to use. Be careful, currently the language code must be of the form *xx* and not *xx-XX*. To see the list of available language codes, take a look at : `mymoney/static/bower_components/bootstrap-datepicker/js/locales/bootstrap-datepicke`

For example, for a French minify JS file, you should execute:

```
gulp js --lang=fr-FR --lang_bt_cal=fr-FR --lang_bt_dp=fr
```

Note: Seems too much verbose to specify 3 arguments for languages but unfortunetly, none of them used the same...

Note: Only *French* internationalisation/translations are supported for now. But any contributions are welcome!

1.1.4 Demo

To have a quick look, you could generate some data with the following commands:

```
./manage.py demo
```

You can also clear any data relatives to the project's models with:

```
./manage.py demo --purge
```

1.1.5 Tests

Whichever method is used, you must create a setting file for testing. Copy `mymoney/settings/test.dist` to `mymoney/settings/test.py` and edit it:

```
cp mymoney/settings/test.dist mymoney/settings/test.py
```

Tox

You can use Tox. At the project root directory without virtualenv, just execute:

```
tox
```

Behind the scenes, it runs several *testenv* for:

- flake8
- isort
- Sphinx
- test suites with coverage and report

Manually

1. install dependencies:

```
pip install -r requirements/test.txt
```

2. then execute tests:

```
./manage.py test --settings=mymoney.settings.test mymoney
```

1.2 Users

1.2.1 Create a basic user

To don't mix superuser and basic user permissions, you will need to create and use **only** a *basic* user. Otherwise, if you use the superuser account on front-office and try to create a bank account, you won't see your own account in the owner list.

Note: This is intentional : a basic user cannot add super user or staff user as an owner of a bank account.

1. create the superuser if not already:

```
./manage.py createsuperuser
```

2. connect to the Django backoffice in order to create a user account.

Warning: Don't forget to assign any permissions required (i.e: beginning with *bank*).

1.2.2 Permissions

Each permissions are derived from the default Django model (add, change, delete). However, here is additional permissions:

- administer owners: allow user to manager owners of a bank account

1.2.3 Anonymous user

Because being authenticated is required, an anonymous user could **only** access the `/login` url (`LOGIN_URL`) or back-office (`ADMIN_BASE_URL`). Any attempt as an anonymous user to access an another url would redirect to the login page.

1.3 Bank accounts

The first thing you should do for the first time is to create a new bank account. After being logged in, you should be on the page `/bank-account/`. You could see *actions links* on the top right corner. Click on *Add*.

Once created, you should see it in the list. Click on it. You should go to the bank account overview page. From this page, you could navigate to:

- edit the bank account
- delete the bank account (through *action links*)
- schedule bank transactions of this bank account
- see statistics of this bank account
- add bank transactions for that bank account

- filter results with a form
- apply some actions on bank transactions

Note: If you have **only one** bank account, you will be redirected to its page after being logged in. Otherwise, you will be redirected on the bank accounts list page at `/bank-account/`.

1.4 Tags

The second step could be to create tags. In the navbar, in *tags* section, click on *My tags* (`/bank-transaction-tag/list/`). You are on the tags overview page. From this page, you could apply the following actions on tags:

- add
- edit
- delete

Note: Owners of the same bank account could also view/use your tags (so even if they are not the owner of the tag). However, a user without relationship (i.e: not being the owner of your bank accounts) can't view/use your tags.

1.5 Schedulers

Even before trying to insert some bank transaction, a better approach would be to create first bank transaction schedulers (for recurring payments for e.g).

On the page of the bank account, click on the link *Schedule* in the menu tab links. You are redirected to the scheduler overview page. On this page, you could:

- see a summary of periodic debit/credit
- add/edit/delete scheduler

1.5.1 Fields

Some fields need more explanations:

Period

For the moment, there are two kinds of periods:

- weekly: clone bank transactions every week for a given date, depending on localization (i.e. first day of the week).
- monthly: clone bank transactions every month for a given date. Don't worry, each month is properly respected : The 2015-01-29 will be 2015-02-28 for the next month.

Recurrence

You can specify how many time the scheduler will be repeat with the *recurrence* field. Leave it empty to be repeated indefinitely. If not infinite, when 0 is reached, the scheduler is automatically deleted.

Date

The *date* is used to be repeated for the next corresponding period. For example, if you have a rent every 10 of the month, you should write a date where day is 10, and month is the **current** month (not the next month), even if the current day is 26 for example.

> **Warning:** Keep in mind that when the background task (cron) would try to clone a bank transaction, it will create it for the **next** date.

Start now

When you create a scheduler, you may be interesting in running it immediately. However be careful, it would create a new bank transaction for the **next** period based on the date/period given. Thus, if you want to create an automatic bank transaction for the current month, the date field must be set for the previous month.

1.6 Bank transactions

Once bank accounts, tags, and schedulers are created, you could now begin to create other bank transactions. You can manage bank transactions from the related bank account page overview:

- add
- edit
- delete

On the bank account page, you can filter bank transactions. But also apply some actions in bulk by checking the corresponding boxes:

- reconciled selected bank transactions
- unreconciled selected bank transactions
- delete

Bank transactions alter the bank account balance when they are:

- created
- updated
- deleted

If you don't want to apply these alterations, you may set the bank transaction's status to *inactive*. See *Status*.

1.6.1 Fields

Some fields need more explanations:

Status

Each statuses may have the following actions:

Status	Alter balance	Statistics
active	Yes	Yes
ignored	Yes	No
inactive	No	No

Indeed, bank transactions could:

- alter the bank account balance. Thus, they are used for the total of the balance (future, current, reconciled)
- being used for statistics (ratio and trendtime)

Active is the default status. *Ignored* could be used for internal transfer for example. You may need it if you want to alter the bank account balance but don't want to pollute your statistics with. *Inactive* exists for purpose only. I didn't find any use cases for it now. May be changed/removed in the future.

Reconcile flag

This is a marker which indicates whether it is synchronous with your real bank account. It is usefull to anticipate expenses for example. You can see a quick summary of the reconciled balance as well as the total reconciled balance. On the contrary, you can also see the balance from a given bank transaction or the future total balance.

1.7 Statistics

You can check statistics for a given bank account. To do so, go on the bank account page and click on the menu tab item *Statistics*. You can then switch between two view modes:

- ratio report: based on tags
- trendtime report: based on dates

Each report provides some basic filters and a quick summary. Each summary has more details when you click on a targeted item.

1.7.1 Ratio

Filter type: sum or single

Ratio statistics display results grouped by tags. To have relevant percentage, numbers compared must be of the same sign (positive or negative). That's why there is two *sub-types*:

- expenses
- wages

For an expense search with *single* mode, **each negative bank transactions** are used. For an expenses search with *sum* mode, **each negative sum** of tags are used.

For example, imagine you do some shopping for $100. Unfortunetely, you discover that an item isn't good. You return it and get a refund of $25. With *single* type filter, the amount used would be -$100. Whereas with *sum* type filter, the amount used would be -100 + 25 = -$75.

Most of the time, the desired mode should be *sum*, that's why this is the default mode.

1.8 Back-office

Anytime, remember that if you need to bypass the default UI, you could still use the default Django back-office, reachable at `/admin` (by default) with a staff or superuser account.

Note: You can modify the setting `ADMIN_BASE_URL` to an other value than *admin* for obvious security reason.

Warning: With the production setting template, the `ADMIN_BASE_URL` is intentionally set to an empty value to throw a CRITICAL check messages when the following command is run `./manage.py check --deploy`.