I hate money

Release 1.0

Nov 01, 2017

Contents

«I hate money» is a web application made to ease shared budget management. It keeps track of who bought what, when, and for whom; and helps to settle the bills.

CHAPTER 1

Table of content

1.1 Installation

1.1.1 Requirements

«Ihatemoney» depends on:

- **Python**: either 2.7, 3.4, 3.5, 3.6 will work.
- **A Backend**: to choose among MySQL, PostgreSQL, SQLite or Memory.
- **Virtualenv** (recommended): *virtualenv* package under Debian/Ubuntu.

We recommend to use virtualenv but it will work without if you prefer.

If wondering about the backend, SQLite is the simplest and will work fine for most small to medium setups.

Note: If curious, source config templates can be found in the project git repository.

1.1.2 Prepare virtualenv (recommended)

Choose an installation path, here */home/john/ihatemoney*.

Create a virtualenv:

```
virtualenv  -p /usr/bin/python3 /home/john/ihatemoney
```

Activate the virtualenv:

```
source /home/john/ihatemoney/bin/activate
```

Note: You will have to re-issue that `source` command if you open a new terminal.

1.1.3 Install

Install the latest release with pip:

```
pip install ihatemoney
```

1.1.4 Test it

Once installed, you can start a test server:

```
ihatemoney runserver
```

And point your browser at http://localhost:5000.

1.1.5 Deploy it

Now, if you want to deploy it on your own server, you have many options. Three of them are documented at the moment.

Of course, if you want to contribute another configuration, feel free to open a pull-request against this repository!

Whatever your installation option is…

1. Initialize the ihatemoney directories:

   ```
   mkdir /etc/ihatemoney /var/lib/ihatemoney
   ```

2. Generate settings:

   ```
   ihatemoney generate-config ihatemoney.cfg > /etc/ihatemoney/ihatemoney.cfg
   chmod 740 /etc/ihatemoney/ihatemoney.cfg
   ```

You probably want to adjust */etc/ihatemoney/ihatemoney.cfg* contents, you may do it later, see *Configuration*.

With Apache and mod_wsgi

1. Fix permissions (considering *www-data* is the user running apache):

   ```
   chgrp www-data /etc/ihatemoney/ihatemoney.cfg
   chown www-data /var/lib/ihatemoney
   ```

2. Install Apache and mod_wsgi - libapache2-mod-wsgi(-py3) for Debian based and mod_wsgi for RedHat based distributions -
3. Create an Apache virtual host, the command `ihatemoney generate-config apache-vhost.conf` will output a good starting point (read and adapt it)
4. Activate the virtual host if needed and restart Apache

With Nginx, Gunicorn and Supervisord

1. Create a dedicated unix user (here called *ihatemoney*), required dirs, and fix permissions:

   ```
   useradd ihatemoney
   chown ihatemoney /var/lib/ihatemoney/
   chgrp ihatemoney /etc/ihatemoney/ihatemoney.cfg
   ```

2. Create gunicorn config file

   ```
   ihatemoney generate-config gunicorn.conf.py > /etc/ihatemoney/gunicorn.conf.py
   ```

3. Create supervisor config file

   ```
   ihatemoney generate-config supervisord.conf > /etc/supervisor/conf.d/ihatemoney.
   ↪conf
   ```

4. Copy (and adapt) output of `ihatemoney generate-config conf/nginx.conf` with your nginx vhosts[1]
5. Reload both nginx and supervisord. It should be working ;)

With Docker

Build the image:

```
docker build -t ihatemoney --build-arg INSTALL_FROM_PYPI=True .
```

Start a daemonized Ihatemoney container:

```
docker run -d -p 8000:8000 ihatemoney
```

Ihatemoney is now available on http://localhost:8000.

All Ihatemoney settings can be passed with `-e` parameters e.g. with a secure `SECRET_KEY`, an external mail server and an external database:

```
docker run -d -p 8000:8000 \
-e SECRET_KEY="supersecure" \
-e SQLALCHEMY_DATABASE_URI="mysql+pymysql://user:pass@172.17.0.5/ihm" \
-e MAIL_SERVER=smtp.gmail.com \
-e MAIL_PORT=465 \
-e MAIL_USERNAME=your-email@gmail.com \
-e MAIL_PASSWORD=your-password \
-e MAIL_USE_SSL=True \
ihatemoney
```

A volume can also be specified to persist the default database file:

```
docker run -d -p 8000:8000 -v /host/path/to/database:/database ihatemoney
```

The following gunicorn parameters are also available:

```
GUNICORN_NUM_WORKERS (default: 3)
```

[1] typically, */etc/nginx/conf.d/* or */etc/nginx/sites-available*, depending on your distribution.

1.1.6 Configuration

ihatemoney relies on a configuration file. If you run the application for the first time, you will need to take a few moments to configure the application properly.

Defaults given here, are those for development mode. To know defaults on your deployed instance, simply look at your *ihatemoney.cfg*.

Production values are recommended values for use in production.

Setting name	Default	What does it do?
SQLALCHEMY_DATABASE_URI	`sqlite:///tmp/ihatemoney.db`	Specifies the type of backend to use and its location. More information on the format used can be found on the SQLAlchemy documentation. **Production value:** Set it to some path on your disk. Typically `sqlite:///home/ihatemoney/ihatemoney.db`. Do *not* store it under `/tmp` as this folder is cleared at each boot.
SECRET_KEY	`tralala`	The secret key used to encrypt the cookies. **Production value:** *ihatemoney conf-example ihatemoney.cfg* sets it to something random, which is good.
MAIL_DEFAULT_SENDER	`("Budget manager", "budget@notmyidea.org")`	A python tuple describing the name and email adress to use when sending emails. **Production value:** Any tuple you want.
ACTIVATE_DEMO_PROJECT	`True`	If set to *True*, a demo project will be available on the frontpage. **Production value: Usually, you will want to set it to** instance.
ADMIN_PASSWORD	`""`	Hashed password to access protected endpoints. If left empty, all administrative tasks are disabled. **Production value:** To generate the proper password HASH, use `ihatemoney generate_password_hash` and copy the output into the value of *ADMIN_PASSWORD*.
ALLOW_PUBLIC_PROJECT_CREATION	`True`	If set to `True`, everyone can create a project without entering the admin password If set to `False`, the password needs to be entered (and as such, defined in the settings).
ACTIVATE_ADMIN_DASHBOARD	`False`	If set to *True*, the dashboard will become accessible entering the admin password If set to *True*, a non empty ADMIN_PASSWORD needs to be set
APPLICATION_ROOT	`""`	If empty, ihatemoney will be served at domain root (e.g: *http://domain.tld*), if set to `"foo"`, it will be served from a "folder" (e.g: *http://domain.tld/foo*)

Using an alternate settings path

You can put your settings file where you want, and pass its path to the application using the `IHATEMONEY_SETTINGS_FILE_PATH` environment variable.

e.g.:

```
$ export IHATEMONEY_SETTINGS_FILE_PATH="/path/to/your/conf/file.cfg"
```

1.2 The REST API

All of what's possible to do with the website is also possible via a web API. This document explains how the API is organized and how you can query it.

By default, the API talks JSON. There is no other way to speak with it currently.

1.2.1 Overall organisation

You can access three different things: projects, members and bills. You can also get the balance for a project.

For the examples, I'm using curl, feel free to use whatever you want to do the same thing, curl is not a requirement.

Authentication

To interact with bills and members, and to do something else than creating a project, you need to be authenticated. The only way to authenticate yourself currently is using the "basic" HTTP authentication.

For instance, here is how to see the what's in a project, using curl:

```
$ curl --basic -u demo:demo https://ihatemoney.org/api/projects/demo
```

Projects

You can't list projects, for security reasons. But you can create, update and delete one directly from the API.

The URLs are */api/projects* and */api/projects/<identifier>*.

Creating a project

A project needs the following arguments:

- *name*: The project name (string)
- *id*: the project identifier (string without special chars or spaces)
- *password*: the project password / secret code (string)
- *contact_email*: the contact email

```
$ curl -X POST https://ihatemoney.org/api/projects \
-d 'name=yay&id=yay&password=yay&contact_email=yay@notmyidea.org'
"yay"
```

As you can see, the API retuns the identifier of the project

Getting information about the project

Getting information about the project:

```
$ curl --basic -u demo:demo https://ihatemoney.org/api/projects/demo
{
    "name": "demonstration",
    "contact_email": "demo@notmyidea.org",
    "password": "demo",
    "id": "demo",
    "active_members": [{"activated": true, "id": 31, "name": "Arnaud"},
                        {"activated": true, "id": 32, "name": "Alexis"},
                        {"activated": true, "id": 33, "name": "Olivier"},
                        {"activated": true, "id": 34, "name": "Fred"}],
    "members": [{"activated": true, "id": 31, "name": "Arnaud"},
                {"activated": true, "id": 32, "name": "Alexis"},
                {"activated": true, "id": 33, "name": "Olivier"},
                {"activated": true, "id": 34, "name": "Fred"}],
}
```

Updating a project

Updating a project is done with the *PUT* verb:

```
$ curl --basic -u yay:yay -X PUT\
https://ihatemoney.org/api/projects/yay -d\
'name=yay&id=yay&password=yay&contact_email=youpi@notmyidea.org'
```

Deleting a project

Just send a DELETE request ont the project URI

```
$ curl --basic -u demo:demo -X DELETE https://ihatemoney.org/api/projects/demo
```

Members

You can get all the members with a *GET* on */api/projects/<id>/members*:

```
$ curl --basic -u demo:demo https://ihatemoney.org/api/projects/demo/members\
[{"activated": true, "id": 31, "name": "Arnaud"},
 {"activated": true, "id": 32, "name": "Alexis"},
 {"activated": true, "id": 33, "name": "Olivier"},
 {"activated": true, "id": 34, "name": "Fred"}]
```

Add a member with a *POST* request on */api/projects/<id>/members*:

```
$ curl --basic -u demo:demo -X POST\
https://ihatemoney.org/api/projects/demo/members -d 'name=tatayoyo'
35
```

You can also *PUT* a new version of a member (changing its name):

```
$ curl --basic -u demo:demo -X PUT\
https://ihatemoney.org/api/projects/demo/members/36\
-d 'name=yeaaaaah'
{"activated": true, "id": 36, "name": "yeaaaaah"}
```

Delete a member with a *DELETE* request on */api/projects/<id>/members/<member-id>*:

```
$ curl --basic -u demo:demo -X DELETE\
https://ihatemoney.org/api/projects/demo/members/35
"OK
```

Bills

You can get the list of bills by doing a *GET* on */api/projects/<id>/bills*

```
$ curl --basic -u demo:demo https://ihatemoney.org/api/projects/demo/bills
```

Add a bill with a *POST* query on */api/projects/<id>/bills*. you need the following params:

- *date*: the date of the bill; defaults to current date if not provided. (yy-mm-dd)
- *what*: what have been payed
- *payer*: by who ? (id)
- *payed_for*: for who ? (id, repeat the parameter to set multiple id)
- *amount*: amount payed

Returns the id of the created bill

```
$ curl --basic -u demo:demo -X POST\
https://ihatemoney.org/api/projects/demo/bills\
-d "date=2011-09-10&what=raclette&payer=31&payed_for=31&amount=200"
80
```

You can also *PUT* a new version of the bill at */api/projects/<id>/bills/<bill-id>*:

```
$ curl --basic -u demo:demo -X PUT\
https://ihatemoney.org/api/projects/demo/bills/80\
-d "date=2011-09-10&what=raclette&payer=31&payed_for=31&amount=250"
80
```

And you can of course *DELETE* them at */api/projects/<id>/bills/<bill-id>*:

```
$ curl --basic -u demo:demo -X DELETE\
https://ihatemoney.org/api/projects/demo/bills/80\
"OK"
```

1.3 Contributing

1.3.1 Setup a dev environment

You must develop on top of the git master branch:

```
git clone https://github.com/spiral-project/ihatemoney.git
```

Then you need to build your dev environments. Choose your way…

The quick way

If System *Requirements* are fulfilled, you can just issue:

```
make serve
```

It will setup a virtualenv, install dependencies, and run the test server.

The hard way

Alternatively, you can also use the *requirements.txt* file to install the dependencies yourself. That would be:

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

And then run the application:

```
cd ihatemoney
python run.py
```

Accessing dev server

In any case, you can point your browser at http://localhost:5000. It's as simple as that!

Updating

In case you want to update to newer versions (from git), you can just run the "update" command:

```
make update
```

Useful settings

It is better to actually turn the debugging mode on when you're developing. You can create a `settings.cfg` file, with the following content:

```
DEBUG = True
SQLACHEMY_ECHO = DEBUG
```

You can also set the *TESTING* flag to *True* so no mails are sent (and no exception is raised) while you're on development mode. Then before running the application, declare its path with

```
$ export IHATEMONEY_SETTINGS_FILE_PATH="$(pwd)/settings.cfg"
```

1.3.2 How to contribute

You would like to contribute? First, thanks a bunch! This project is a small project with just a few people behind it, so any help is appreciated!

There are different ways to help us, regarding if you are a designer, a developer or an user.

As a developer

If you want to contribute code, you can write it and then issue a pull request on github. Please, think about updating and running the tests before asking for a pull request as it will help us to maintain the code clean and running.

To do so:

```
$ make test
```

As a designer / Front-end developer

Feel free to provide us mockups or to involve yourself into the discussions hapenning on the github issue tracker. All ideas are welcome. Of course, if you know how to implement them, feel free to fork and make a pull request.

End-user

You are using the application and found a bug? You have some ideas about how to improve the project? Please tell us by filling a new issue. Or, if you prefer, you can send me an email to *alexis@notmyidea.org* and I will update the issue tracker with your feedback.

Thanks again!

1.3.3 How to build the documentation ?

The documentation is using sphinx and its source is located inside the docs folder.

Install doc dependencies (within the virtualenv, if any):

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

And to produce html doc in *docs/_output* folder:

```
cd docs/
make html
```

1.3.4 How to release?

In order to prepare a new release, we are following the following steps:

- Merge remaining pull requests;
- Update `CHANGELOG.rst` with the last changes;
- Update `CONTRIBUTORS`;
- Update known good versions of dependencies in `requirements.txt` with this command (from inside the venv):

```
$ pip freeze | grep -v -- '-e' > requirements.txt
```

Once this is done, use the “release” instruction:

```
$ make release
```

And the new version should be published on PyPI.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search