I hate money

Release 5.0

Oct 19, 2021

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.

I hate money is written in python, using the flask framework. It's developed with ease of use in mind, and is trying to keep things simple. Hope you (will) like it!

CHAPTER 1

Table of contents

1.1 Installation

We lack some knowledge about packaging to make Ihatemoney installable on mainstream Linux distributions. If you want to give us a hand on the topic, please check-out the issue about debian packaging.

If you are using Yunohost (a server operating system aiming to make self-hosting accessible to anyone), you can use the Ihatemoney package.

Otherwise, follow these instructions to install it manually:

1.1.1 Requirements

«Ihatemoney» depends on:

- **Python**: version 3.6 to 3.9 included will work.
- **A Backend**: to choose among SQLite, PostgreSQL, MariaDB (>= 10.3.2) or Memory.
- **Virtual environment** (recommended): *python3-venv* package under Debian/Ubuntu.

We recommend to use virtual environment 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 virtual environment (recommended)

Choose an installation path, here the current user's home directory (~).

Create a virtual environment:

```
python3 -m venv ~/ihatemoney
cd ~/ihatemoney
```

Activate the virtual environment:

```
source 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
```

Warning: The current release of ihatemoney (4.1.5) does not work with SQLAlchemy 1.4. The dependency will be fixed in the next version, but in the meantime you can work around the issue with: `pip install 'SQLAlchemy>=1.3,<1.4'`.

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 Configure database with MariaDB (optional)

Note: Only required if you use MariaDB. Make sure to use MariaDB 10.3.2 or newer.

1. Install PyMySQL dependencies. On Debian or Ubuntu, that would be:

   ```
   apt install python3-dev libssl-dev
   ```

2. Install PyMySQL (within your virtual environment):

   ```
   pip install 'PyMySQL>=0.9,<1.1'
   ```

3. Create an empty database and a database user
4. Configure *SQLALCHEMY_DATABASE_URI* accordingly

1.1.6 Configure database with PostgreSQL (optional)

Note: Only required if you use Postgresql.

1. Install python driver for PostgreSQL (from within your virtual environment):

```
pip install psycopg2
```

2. Create the users and tables. On the command line, this looks like:

```
sudo -u postgres psql
postgres=# create database mydb;
postgres=# create user myuser with encrypted password 'mypass';
postgres=# grant all privileges on database mydb to myuser;
```

3. Configure *SQLALCHEMY_DATABASE_URI* accordingly.

1.1.7 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/systemd

Install Gunicorn:

```
pip install gunicorn
```

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. Setup Supervisord or systemd

   - To use Supervisord, create supervisor config file

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

   - To use systemd services, create `ihatemoney.service` in[2]:

     ```
     [Unit]
     Description=I hate money
     Requires=network.target postgresql.service
     After=network.target postgresql.service

     [Service]
     Type=simple
     User=ihatemoney
     ExecStart=%h/ihatemoney/bin/gunicorn -c /etc/ihatemoney/gunicorn.conf.py␣
     ↪ihatemoney.wsgi:application
     SyslogIdentifier=ihatemoney

     [Install]
     WantedBy=multi-user.target
     ```

     Obviously, adapt the `ExecStart` path for your installation folder.

     If you use SQLite as database: remove mentions of `postgresql.service` in `ihatemoney.service`. If you use MariaDB as database: replace mentions of `postgresql.service` by `mariadb.service` in `ihatemoney.service`.

     Then reload systemd, enable and start `ihatemoney`:

     ```
     systemctl daemon-reload
     systemctl enable ihatemoney.service
     systemctl start ihatemoney.service
     ```

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

[2] `/etc/systemd/system/ihatemoney.service` path may change depending on your distribution.
[1] typically, */etc/nginx/conf.d/* or */etc/nginx/sites-available*, depending on your distribution.

With Docker

Build the image:

```
docker build -t ihatemoney .
```

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
```

To enable the Admin dashboard, first generate a hashed password with:

```
docker run -it --rm --entrypoint ihatemoney ihatemoney generate_password_hash
```

At the prompt, enter a password to use for the admin dashboard. The command will print the hashed password string.

Add these additional environment variables to the docker run invocation:

```
-e ACTIVATE_ADMIN_DASHBOARD=True \
-e ADMIN_PASSWORD=<hashed_password_string> \
```

Additional gunicorn parameters can be passed using the docker `CMD` parameter. For example, use the following command to add more gunicorn workers:

```
docker run -d -p 8000:8000 ihatemoney -w 3
```

1.2 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.

The default values given here are those for the development mode. To know defaults on your deployed instance, simply look at your `ihatemoney.cfg` file.

"Production values" are the recommended values for use in production.

1.2.1 Configuration files

By default, Ihatemoney loads its configuration from `/etc/ihatemoney/ihatemoney.cfg`.

If you need to load the configuration from a custom path, you can define the `IHATEMONEY_SETTINGS_FILE_PATH` environment variable with the path to the configuration file. For instance

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

The path should be absolute. A relative path will be interpreted as being inside `/etc/ihatemoney/`.

1.2.2 *SQLALCHEMY_DATABASE_URI*

Specifies the type of backend to use and its location. More information on the format used can be found on the SQLAlchemy documentation.

- **Default value:** `sqlite:///tmp/ihatemoney.db`
- **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.

For example, if you're using MariaDB, use a configuration similar to the following:

```
SQLALCHEMY_DATABASE_URI = 'mysql+pymysql://user:pass@localhost/mydb'
```

If you're using PostgreSQL, your client must use utf8. Unfortunately, PostgreSQL default is to use ASCII. Either change your client settings, or specify the encoding by appending `?client_encoding=utf8` to the connection string. This will look like:

```
SQLALCHEMY_DATABASE_URI = 'postgresql://myuser:mypass@localhost/mydb?client_
↪encoding=utf8'
```

1.2.3 *SECRET_KEY*

The secret key used to encrypt cookies and generate secure tokens. They are used to authenticate access to projects, both through the web interface and through the API.

As such, you should never use a predictible secret key: an attacker with the knowledge of the secret key could easily access any project and bypass the private code verification.

- **Production value:** *ihatemoney conf-example ihatemoney.cfg* sets it to something random, which is good.

1.2.4 *SESSION_COOKIE_SECURE*

A boolean that controls whether the session cookie will be marked "secure". If this is the case, browsers will refuse to send the session cookie over plain HTTP.

- **Default value:** `True`
- **Production value:** `True` if you run your service over HTTPS, `False` if you run your service over plain HTTP.

Note: this setting is actually interpreted by Flask, see the Flask documentation for details.

1.2.5 *MAIL_DEFAULT_SENDER*

A python tuple describing the name and email address to use when sending emails.

- **Default value:** `("Budget manager", "budget@notmyidea.org")`
- **Production value:** Any tuple you want.

1.2.6 *ACTIVATE_DEMO_PROJECT*

If set to *True*, a demo project will be available on the frontpage.

- **Default value:** `True`
- **Production value:** Usually, you will want to set it to `False` for a private instance.

1.2.7 *ADMIN_PASSWORD*

Hashed password to access protected endpoints. If left empty, all administrative tasks are disabled.

- **Default value:** `""` (empty string)
- **Production value:** To generate the proper password HASH, use `ihatemoney generate_password_hash` and copy the output into the value of *ADMIN_PASSWORD*.

1.2.8 *ALLOW_PUBLIC_PROJECT_CREATION*

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).

- **Default value:** : `True`.

1.2.9 *ACTIVATE_ADMIN_DASHBOARD*

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.

- **Default value**: `False`

1.2.10 *APPLICATION_ROOT*

If empty, ihatemoney will be served at domain root (e.g: *http://domain.tld*), if set to `"somestring"`, it will be served from a "folder" (e.g: *http://domain.tld/somestring*).

- **Default value:** `""` (empty string)

1.2.11 *BABEL_DEFAULT_TIMEZONE*

The timezone that will be used to convert date and time when displaying them to the user (all times are always stored in UTC internally). If not set, it will default to the timezone configured on the Operating System of the server running ihatemoney, which may or may not be what you want.

- **Default value:** *unset* (use the timezone of the server Operating System)

- **Production value:** Set to the timezone of your expected users, with a format such as `"Europe/Paris"`. See **'this list of TZ database names'_** for a complete list.

Note: this setting is actually interpreted by Flask-Babel, see the Flask-Babel guide for formatting dates for details.

1.2.12 *ENABLE_CAPTCHA*

It is possible to add a simple captcha in order to filter out spammer bots on the form creation. In order to do so, you just have to set *ENABLE_CAPTCHA = True*.

1.2.13 Configuring emails sending

By default, Ihatemoney sends emails using a local SMTP server, but it's possible to configure it to act differently, thanks to the great Flask-Mail project

- **MAIL_SERVER** : default **'localhost'**
- **MAIL_PORT** : default **25**
- **MAIL_USE_TLS** : default **False**
- **MAIL_USE_SSL** : default **False**
- **MAIL_DEBUG** : default **app.debug**
- **MAIL_USERNAME** : default **None**
- **MAIL_PASSWORD** : default **None**
- **DEFAULT_MAIL_SENDER** : default **None**

1.3 Upgrading

We keep a ChangeLog. Read it before upgrading.

Ihatemoney follows semantic versioning. So minor/patch upgrades can be done blindly.

1.3.1 General procedure

(sufficient for minor/patch upgrades)

1. From the virtual environment (if any):

```
pip install -U ihatemoney
```

2. Restart *supervisor*, or *Apache*, depending on your setup.

You may also want to set new configuration variables (if any). They are mentioned in the ChangeLog, but this is **not required for minor/patch upgrades**, a safe default will be used automatically.

1.3.2 Version-specific instructions

(must read for major upgrades)

When upgrading from a major version to another, you **must** follow special instructions:

4.x → 5.x

Switch to a supported version of Python

Note: If you are already using Python 3.6, you can skip this section, no special action is required.

If you were running IHateMoney using Python < 3.6, you must, **before** upgrading:

1. Ensure to have a Python 3.6 available on your system
2. Rebuild your virtual environment (if any). It will *not* alter your database nor configuration. For example, if your virtual environment is in */home/john/ihatemoney/*:

```
rm -rf /home/john/ihatemoney
pyhton3 -m venv /home/john/ihatemoney
source /home/john/ihatemoney/bin/activate
```

   You might need to `pip install` additional dependencies if you are using one or several of the following deployment options :

   - Gunicorn (Nginx)
   - MariaDB
   - PostgreSQL

If so, pick the `pip` commands to use in the relevant section(s) of *Installation*.

Then follow *General procedure* from step 1. in order to complete the update.

Disable session cookie security if running over plain HTTP

Note: If you are running Ihatemoney over HTTPS, no special action is required.

Session cookies are now marked "secure" by default to increase security.

If you run Ihatemoney over plain HTTP, you need to explicitly disable this security feature by setting `SESSION_COOKIE_SECURE` to `False`, see *Configuration*.

Switch to MariaDB >= 10.3.2 instead of MySQL

Note: If you are using SQLite or PostgreSQL, you can skip this section, no special action is required.

If you were running IHateMoney with MySQL, you must switch to MariaDB. MySQL is no longer a supported database option.

In addition, the minimum supported version of MariaDB is 10.3.2. See this MySQL / MariaDB issue for details.

To upgrade:

1. Ensure you have a MariaDB server installed and configured, and that its version is at least 10.3.2.
2. Copy your database from MySQL to MariaDB.

3. Ensure that IHateMoney is correctly configured to use your MariaDB database, see *Configuration*.

Then follow *General procedure* from step 1. in order to complete the update.

2.x → 3.x

Sentry support has been removed. Sorry if you used it.

Appart from that, *General procedure* applies.

1.x → 2.x

Switch from git installation to pip installation

The recommended installation method is now using *pip*. Git is now intended for development only.

Warning: Be extra careful to not remove your sqlite database nor your settings file, if they are stored inside the cloned folder.

1. Delete the cloned folder

Note: If you are using a virtual environment, then the following commands should be run inside it (see *Prepare virtual environment (recommended)*).

2. Install ihatemoney with pip:

```
pip install ihatemoney
```

3. Fix your configuration file (paths *have* changed), depending on the software you use in your setup:
   - **gunicorn**: `ihatemoney generate-config gunicorn.conf.py` (nothing critical changed, keeping your old config might be fine)
   - **supervisor** : `ihatemoney generate-config supervisord.conf` (mind the `command=` line)
   - **apache**: `ihatemoney generate-config apache-vhost.conf` (mind the `WSGIDaemonProcess`, `WSGIScriptAlias` and `Alias` lines)
4. Restart *Apache* or *Supervisor*, depending on your setup.

Upgrade ADMIN_PASSWORD to its hashed form

Note: Not required if you are not using the ADMIN_PASSWORD feature.

`ihatemoney generate_password_hash` will do the hashing job for you, just put its result in the `ADMIN_PASSWORD` var from your *ihatemoney.cfg* and restart *apache* or the *supervisor* job.

1.4 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.

The main supported data format is JSON. When using POST or PUT, you can either pass data encoded in JSON or in `application/x-www-form-urlencoded` format.

1.4.1 Overall organisation

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

The examples here are 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 for any action other than creating a new project, you need to be authenticated. The simplest way to authenticate is to use "basic" HTTP authentication with the project ID and private code.

For instance, to obtain information about a project, using curl:

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

It is also possible to generate a token, and then use it later to authenticate instead of basic auth. For instance, start by generating the token (of course, you need to authenticate):

```
$ curl --basic -u demo:demo https://ihatemoney.org/api/projects/demo/token
{"token": "WyJ0ZXN0Il0.Rt04fNMmxp9YslCRq8hB6jE9s1Q"}
```

Make sure to store this token securely: it allows full access to the project. For instance, use it to obtain information about the project (replace PROJECT_TOKEN with the actual token):

```
$ curl --oauth2-bearer "PROJECT_TOKEN" https://ihatemoney.org/api/projects/demo
```

This works by sending the token in the Authorization header, so doing it "manually" with curl looks like:

```
$ curl --header "Authorization: Bearer PROJECT_TOKEN" https://ihatemoney.org/api/
↪projects/demo
```

This token can also be used to authenticate for a project on the web interface, which can be useful to generate invitation links. You would simply create an URL of the form:

```
https://ihatemoney.org/demo/join/PROJECT_TOKEN
```

Such a link grants full access to the project associated with the token.

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 (string)

Optional arguments:

- `default_currency`: the default currency to use for a multi-currency project, in ISO 4217 format. Bills are converted to this currency for operations like balance or statistics. Default value: `XXX` (no currency).

```
$ 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 returns 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
{
    "id": "demo",
    "name": "demonstration",
    "contact_email": "demo@notmyidea.org",
    "default_currency": "XXX",
    "members": [{"id": 11515, "name": "f", "weight": 1.0, "activated": true, "balance
↪": 0},
                {"id": 11531, "name": "g", "weight": 1.0, "activated": true, "balance
↪": 0},
                {"id": 11532, "name": "peter", "weight": 1.0, "activated": true,
↪"balance": 5.0},
                {"id": 11558, "name": "Monkey", "weight": 1.0, "activated": true,
↪"balance": 0},
                {"id": 11559, "name": "GG", "weight": 1.0, "activated": true, "balance
↪": -5.0}]
}
```

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\
[{"weight": 1, "activated": true, "id": 31, "name": "Arnaud"},
 {"weight": 1, "activated": true, "id": 32, "name": "Alexis"},
 {"weight": 1, "activated": true, "id": 33, "name": "Olivier"},
 {"weight": 1, "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", "weight": 1}
```

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
```

Or get a specific bill by ID:

```
$ curl --basic -u demo:demo https://ihatemoney.org/api/projects/demo/bills/42
{
  "id": 42,
  "payer_id": 11,
  "owers": [
    {
      "id": 22,
      "name": "Alexis",
      "weight": 1,
      "activated": true
    }
```

(continues on next page)

(continued from previous page)

```
  ],
  "amount": 100,
  "date": "2020-12-24",
  "creation_date": "2021-01-13",
  "what": "Raclette du nouvel an",
  "external_link": "",
  "original_currency": "XXX",
  "converted_amount": 100
}
```

`amount` is expressed in the `original_currency` of the bill, while `converted_amount` is expressed in the project `default_currency`. Here, they are the same.

Add a bill with a `POST` query on `/api/projects/<id>/bills`. You need the following required parameters:

- `what`: what has been paid (string)
- `payer`: paid by who? (id)
- `payed_for`: for who ? (id). To set multiple id, simply pass the parameter multiple times (x-www-form-urlencoded) or pass a list of id (JSON).
- `amount`: amount payed (float)

And optional parameters:

- `date`: the date of the bill (`yyyy-mm-dd` format). Defaults to current date if not provided.
- `original_currency`: the currency in which `amount` has been paid (ISO 4217 code). Only makes sense for a project with currencies. Defaults to the project `default_currency`.
- `external_link`: an optional URL associated with the bill.

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=1&payed_for=3&payed_for=5&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=1&payed_for=3&payed_for=5&payed_for=1&
↪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"
```

Statistics

You can get some project stats with a `GET` on `/api/projects/<id>/statistics`:

```
$ curl --basic -u demo:demo https://ihatemoney.org/api/projects/demo/statistics
[
    {
        "member": {"activated": true, "id": 1, "name": "alexis", "weight": 1.0},
        "paid": 25.5,
        "spent": 15,
        "balance": 10.5
    },
    {
        "member": {"activated": true, "id": 2, "name": "fred", "weight": 1.0},
        "paid": 5,
        "spent": 15.5,
        "balance": -10.5
    }
]
```

1.5 Security

Ihatemoney does not have user accounts. Instead, authorization is based around shared projects: this is a bit unusual and deserves some explanation about the security model.

First of all, Ihatemoney fundamentally assumes that all members of a project trust each other. Otherwise, you would probably not share expenses in the first place!

That being said, there are a few mechanisms to limit the impact of a malicious member and to manage changes in membership (e.g. ensuring that a previous member can no longer access the project). But these mechanisms don't prevent a malicious member from breaking things in your project!

1.5.1 Security model

A project has three main parameters when it comes to security:

- **project identifier** (equivalent to a "login")
- **private code** (equivalent to a "password")
- **token** (cryptographically derived from the private code)

Somebody with the private code can:

- access the project through the web interface or the API
- add, modify or remove bills
- view project history
- change basic settings of the project
- change the email address associated to the project
- change the private code of the project

Somebody with the token can manipulate the project through the API to do essentially the same thing:

- access the project
- add, modify or remove bills
- change basic settings of the project

- change the email address associated to the project
- change the private code of the project

The token can also be used to build "invitation links". These links allow to login on the web interface without knowing the private code, see below.

1.5.2 Giving access to a project

There are two main ways to give access to a project to a new person:

- share the project identifier and private code using any out-of-band communication method
- share an invitation link that allows to login on the web interface without knowing the private code

The second method is interesting because it does not reveal the private code. In particular, somebody that is logged-in through the invitation link will not be able to change the private code, because the web interface requires a confirmation of the existing private code to change it. However, a motivated person could extract the token from the invitation link, use it to access the project through the API, and change the private code through the API.

1.5.3 Removing access to a project

If a person should no longer be able to access a project, the only way is to change the private code.

This will also automatically change the token: old invitation links won't work anymore, and anybody with the old token will no longer be able to access the project through the API.

1.5.4 Recovering access to a project

If the private code is no longer known, the creator of the project can still recover access. He/she must have provided an email address when creating the project, and Ihatemoney can send a reset link to this email address (classical "forgot your password" functionality).

Note, however, that somebody with the private code could have changed the email address in the settings at any time.

1.5.5 Recovering lost data

A member can delete or change bills. There is no way to revert such actions for now. However, each project has an history page that lists all actions done on the project. This history can be used to manually correct previous changes.

Note, however, that the history feature is primarily meant to protect against mistakes: a malicious member can easily remove all entries from the history!

The best defense against this kind of issues is... backups! All data for a project can be exported through the settings page or through the API.

1.6 Contributing

1.6.1 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. To get started, please read *Set up a dev environment* and *Contributing as a developer*.

As a designer / Front-end developer

Feel free to provide mockups, or to involve yourself in the discussions happening 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.

As a translator

If you're able to translate Ihatemoney in your own language, head over to the website we use for translations and start translating.

All the heavy lifting will be done automatically, and your strings will eventually be integrated.

Once a language is ready to be integrated, add it to the `SUPPORTED_LANGUAGES` list, in `ihatemoney/default_settings.py`.

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 e-mail to *alexis@notmyidea.org* and I will update the issue tracker with your feedback.

Thanks again!

1.6.2 Set up 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 environment. Choose your way...

The quick way

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

```
make serve
```

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

The hard way

Alternatively, you can use pip to install dependencies yourself. That would be:

```
pip install -e .
```

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
```

Then before running the application, declare its path with

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

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.

In some cases, you may need to disable secure cookies by setting `SESSION_COOKIE_SECURE` to `False`. This is needed if you access your dev server over the network: with the default value of `SESSION_COOKIE_SECURE`, the browser will refuse to send the session cookie over insecure HTTP, so many features of Ihatemoney won't work (project login, language change, etc).

1.6.3 Contributing as a developer

All code contributions should be submitted as Pull Requests on the github project.

Below are some points that you should check to help you prepare your Pull Request.

Running tests

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 run the tests:

```
make test
```

Tests can be edited in `ihatemoney/tests/tests.py`. If some test cases fail because of your changes, first check whether your code correctly handle these cases. If you are confident that your code is correct and that the test cases simply need to be updated to match your changes, update the test cases and send them as part of your pull request.

If you are introducing a new feature, you need to either add tests to existing classes, or add a new class (if your new feature is significantly different from existing code).

Formatting code

We are using black and isort formatters for all the Python files in this project. Be sure to run it locally on your files. To do so, just run:

```
make black isort
```

You can also integrate them with your dev environment (as a *format-on-save* hook, for instance).

Creating database migrations

In case you need to modify the database schema, first make sure that you have an up-to-date database by running the dev server at least once (the quick way or the hard way, see above). The dev server applies all existing migrations when starting up.

You can now update the models in `ihatemoney/models.py`. Then run the following command to create a new database revision file:

```
make create-database-revision
```

If your changes are simple enough, the generated script will be populated with the necessary migrations steps. You can view and edit the generated script, which is useful to review that the expected model changes have been properly detected. Usually the auto-detection works well in most cases, but you can of course edit the script to fix small issues. You could also edit the script to add data migrations.

When you are done with your changes, don't forget to add the migration script to your final git commit!

If the migration script looks completely wrong, remove the script and start again with an empty database. The simplest way is to remove or rename the dev database located at `/tmp/ihatemoney.db`, and run the dev server at least once.

For complex migrations, it is recommended to start from an empty revision file which can be created with the following command:

```
make create-empty-database-revision
```

You then need to write the migration steps yourself.

1.6.4 How to build the documentation ?

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

Install doc dependencies (within the virtual environment, if any):

```
pip install -e .[doc]
```

And to produce a HTML doc in the *docs/_output* folder:

```
cd docs/
make html
```

1.6.5 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 `setup.cfg`
- If needed, recompress assets. It requires zopflipng:

```
make compress-assets
```

- Build the translations:

```
make update-translations
make build-translations
```

Once this is done, use the "release" instruction:

```
make release
```

And the new version should be published on PyPI.

Note: The above command will prompt for version number, handle `CHANGELOG.rst` and `setup.cfg` updates, package creation, pypi upload. It will prompt you before each step to get your consent.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search