Socialhome Documentation

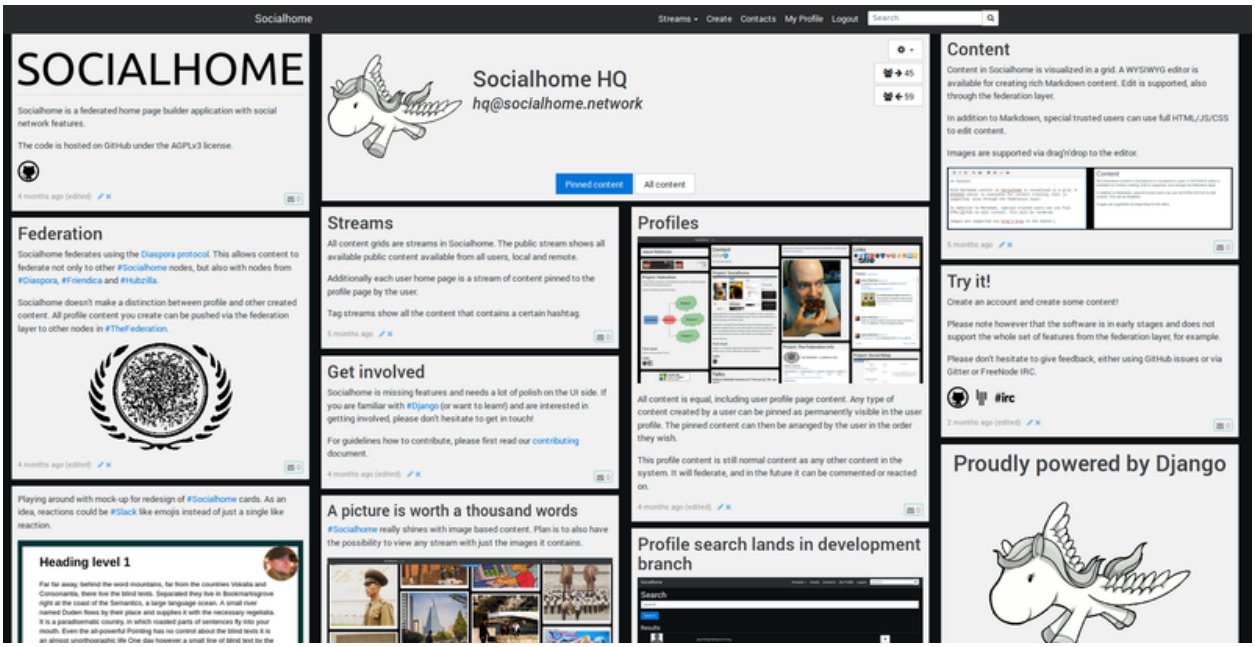
Release 0.8.0-dev

Jason Robinson

Feb 21, 2018

Contents

Welcome to the documentation of Socialhome!



CHAPTER 1

Description

Socialhome is best described as a federated personal profile with social networking functionality. Users can create rich content using Markdown and even HTML/JS/CSS (if set as trusted user). All content can be pinned to the user profile and all content will federate to contacts in the federated social web. Currently federation happens using the Diaspora protocol with future plans to include at least ActivityPub as well. Federating using existing protocols means Socialhome users can interact with tens of thousands of other users.

Please check the official site for more information about features. Naturally, the official site is a Socialhome profile itself.

Official site: https://socialhome.network

CHAPTER 2

Joining

Yes! The official server is open for registrations. Sign up and play around!

Please see the *Community* pages for how to interact with the community.

CHAPTER 3

Installation

Please see the *Installation* pages.

CHAPTER 4

Running an instance

Please see the *Running an instance* pages.

CHAPTER 5

Development

Please see the *Development* pages.

CHAPTER 6

Source code

Socialhome is fully open source, licenced under the AGPLv3 license.

Check the code on GitHub.

CHAPTER 7

Table of contents

7.1 Installation

7.1.1 System requirements

Socialhome requires a Linux server with root access. It is not possible to install Socialhome on a shared server.

Resources

Socialhome isn't particularly heavy, though obviously that depends on the amount of users and connections to remote nodes. In default set up, Socialhome will be running the following:

- uWSGI (~200mb)
- Channels worker (~85mb)
- Daphne (~60mb)
- Circus (~25mb)
- 5x RQ workers (~75mb each == 375mb)
- RQ scheduler (~75mb)

Memory values are taken from a running production instance. This gives a total of 745mb of RAM used by the application. Additionally, you need to allocate for PostgreSQL and Redis. A few gigabytes of available memory should easily be enough to run a node with medium activity.

Disk space will mostly be required for content and image uploads. As an example of database size, 100K content objects takes approx 230mb in the database. Image upload disk requirements depends entirely on the sizes of the uploads.

7.1.2 Guides

See *Install guides*.

7.2 Install guides

These instructions are for a production installation. For development installation instructions, see the *Development* pages.

If you have issues following these instructions, please contact us via *Community*.

Available guides:

- installation-ubuntu
- *Ubuntu (14.04, Ansible)*
- *Other Linuxes or newer Ubuntu using SystemD*

7.2.1 Ubuntu (14.04)

This guide is very opinionated and experienced sysadmins will most likely want to do things differently. This guide will give you a Socialhome production install on uWSGI using an Apache2 web server.

Supported versions

This guide is written for **Ubuntu 14.04** (with Upstart). For a SystemD config file, see *Other Linuxes or newer Ubuntu using SystemD*.

Steps

Fix locales

Ubuntu 14.04 has a problem with locales which could bring problems when installing PostgreSQL. If you have already installed PostgreSQL, you can probably skip this step.

Check these two commands:

```
echo $LANGUAGE
echo $LC_ALL
```

if both of them come out empty, edit the file `/etc/default/locale` and add the two following lines:

```
LANGUAGE="en_US.UTF-8"
LC_ALL="en_US.UTF-8"
```

Save, logout and log back in.

See this post for example for a description of this problem.

Install system packages

```
# Generic packages needed
sudo apt-get install git python-virtualenv python3-setuptools python-dev python3-dev␣
↪build-essential

# PostgreSQL dependencies
sudo apt-get install libpq-dev

# federation dependencies
sudo apt-get install libxml2-dev libxslt-dev lib32z1-dev

# Redis
sudo apt-get install redis-server
```

Install Node.js

Node.js version 6+ is needed for statics management. Install it by following the Node.js install guides.

Install PostgreSQL

If not installed or not using a remote PostgreSQL DB, install the database engine.

```
sudo apt-get install postgresql
```

Create a database and user. Note down password for later.

```
sudo su - postgres
createuser -P socialhome
createdb -O socialhome socialhome
exit
```

Create a local user

It's better to run applications under their own user.

```
sudo adduser socialhome --disabled-login
sudo chmod 750 /home/socialhome

# Add user group to www-data groups so we can protect users home folder
sudo adduser www-data socialhome
```

Set up uWSGI

```
# Create logs path
sudo -u socialhome mkdir /home/socialhome/logs
```

Create the ini file with `/home/socialhome/uwsgi.ini` and add the following contents to it.

```
[uwsgi]
chdir=/home/socialhome/socialhome
module=config.wsgi:application
master=True
pidfile=/tmp/socialhome-master.pid
vacuum=True
max-requests=5000
logto=/home/socialhome/logs/uwsgi-master.log
virtualenv=/home/socialhome/.virtualenvs/socialhome
processes=2
threads=2
enable-threads=True
socket=127.0.0.1:31452/
uid=socialhome
gid=socialhome
harakiri=30
```

Set up Apache

if not already installed, install the Apache2 web server.

```
sudo apt-get install apache2 libapache2-mod-proxy-uwsgi
```

Enable some necessary modules.

```
sudo a2enmod proxy_uwsgi
sudo a2enmod proxy_wstunnel
sudo a2enmod proxy
sudo a2enmod ssl
```

Add an Apache virtualhost file `/etc/apache2/sites-available/socialhome.conf` with the following content, replacing instances of `yourdomain.tld` with your real domain for your Socialhome instance:

```
<VirtualHost *:80>
    ServerName yourdomain.tld
    ServerAlias www.yourdomain.tld
    RedirectPermanent / https://yourdomain.tld/
</VirtualHost>

<VirtualHost *:443>
    ServerName yourdomain.tld
    ServerAlias www.yourdomain.tld
    ServerAdmin webmaster@yourdomain.tld

    Alias /robots.txt /home/socialhome/socialhome/staticfiles/robots.txt
    Alias /favicon.ico /home/socialhome/socialhome/staticfiles/favicon.ico
    Alias /media /home/socialhome/socialhome/socialhome/media

    <Directory /home/socialhome/socialhome/socialhome/media>
        Require all granted
        Options -MultiViews -Indexes
    </Directory>

    ProxyPass /media !
    ProxyPass /ch/ ws://127.0.0.1:23564/ch/
    ProxyPass / uwsgi://127.0.0.1:31452/
```

```
    SSLEngine on
    SSLCertificateFile /etc/letsencrypt/live/yourdomain.tld/cert.pem
    SSLCertificateKeyFile /etc/letsencrypt/live/yourdomain.tld/privkey.pem
    SSLCertificateChainFile /etc/letsencrypt/live/yourdomain.tld/chain.pem
</VirtualHost>
```

Enable Apache virtualhost

```
sudo a2ensite socialhome
```

Get LetsEncrypt certificate

We wouldn't want to run our site without HTTPS. Install Certbot and get an LetsEncrypt certificate.

```
sudo apt-get install software-properties-common
sudo add-apt-repository ppa:certbot/certbot
sudo apt-get update
sudo apt-get install python-certbot-apache
```

Launch Certbot and answer any questions to install the certificates.

```
certbot --apache certonly
```

Now you should be able to restart Apache.

```
sudo service apache2 restart
```

Change to Socialhome user

Change to user `socialhome` for the rest of the guide.

```
sudo su - socialhome
```

Install Virtualenvwrapper

This is the easiest way to manage Python virtualenvs. We also add production Django configuration reference at the same time.

```
pip install --user virtualenvwrapper
```

Add the following lines to your `.bashrc` and reload it via `source ~/.bashrc`.

```
export WORKON_HOME=$HOME/.virtualenvs
source ~/.local/bin/virtualenvwrapper.sh
export DJANGO_SETTINGS_MODULE=config.settings.production
```

Create Python virtualenv

```
mkvirtualenv -p /usr/bin/python3 socialhome
```

The virtualenv is automatically activated. When you need it in the future, just type `workon socialhome`.

Update pip and setuptools

```
pip install -U pip setuptools==30.4
```

Install pip-tools

`pip-tools` is a handy tool to keep environments clean and all dependencies nicely pinned.

```
pip install -U pip-tools
```

Get Socialhome code

```
git clone https://github.com/jaywink/socialhome
cd socialhome
```

Install Python dependencies

We use the `pip-tools` command to ensure dependencies are at the correct versions.

```
pip-sync
```

Create configuration

Create the file `.env` with the following contents, replacing values as needed.

You must change or add the following values:

- Replace `DATABASEPASSWORDHERE` with the database password typed in earlier.
- `DJANGO_SECRET_KEY` must be added. Generate one for example here.
- Place your domain in `DJANGO_ALLOWED_HOSTS` and `SOCIALHOME_DOMAIN`.

```
DATABASE_URL=postgres://socialhome:DATABASEPASSWORDHERE@127.0.0.1:5432/socialhome
DJANGO_SECRET_KEY=
DJANGO_ALLOWED_HOSTS=yourdomain.tld
DJANGO_SECURE_SSL_REDIRECT=True
DJANGO_ACCOUNT_ALLOW_REGISTRATION=True
SOCIALHOME_DOMAIN=yourdomain.tld
SOCIALHOME_HTTPS=True
SOCIALHOME_LOGFILE=/home/socialhome/logs/socialhome.log
```

For further configuration tips, see *Running an instance*.

Make the env file a bit less readable.

```
chmod 0600 .env
```

Configure email sending

Note, email *is* required for signing up. Users will **not** be able to sign up if the instance does not have working email sending. See *DJANGO_EMAIL_BACKEND* on how to configure email sending.

Run migrations

```
python manage.py migrate
```

Install statics

```
npm install
node_modules/.bin/bower install
npm run build
python manage.py collectstatic
```

Search index

The search indexes must be initialized, otherwise there will be an error when trying to use search. Run this command once:

```
python manage.py rebuild_index
```

Any further changes to indexes objects will be maintained automatically from this point onwards. If you ever need to rebuild the index from scratch, use the same command.

Set the correct domain name in Django

Load up the Django shell with `python manage.py shell_plus` and then execute the following, replacing "yourdomain.tld" with your domain and "Socialhome" as the name of your site, assuming you want the name changed:

```
Site.objects.filter(id=1).update(domain="yourdomain.tld", name="Socialhome")
exit
```

Set up Circus

Exit Socialhome user and create Upstart configuration for Circus process manager. Circus is used to control various processes that are needed in addition to the web server. This allows starting one process that will start and maintain a bunch of other processes we need. A configuration file for the processes is provided within the repository.

Create Upstart configuration `/etc/init/socialhome.conf` with the following content:

```
description "Socialhome"
start on runlevel [2345]
stop on runlevel [06]
setuid socialhome
setgid socialhome

respawn

env PYTHONPATH="/home/socialhome/socialhome"
env SOCIALHOME_HOME="/home/socialhome"
env RQWORKER_NUM=5
env VIRTUAL_ENV=/home/socialhome/.virtualenvs/socialhome
env LC_CTYPE=en_US.UTF-8
env LC_ALL=C.UTF-8
env LANG=C.UTF-8
env DJANGO_SETTINGS_MODULE=config.settings.production

chdir /home/socialhome/socialhome

exec /home/socialhome/.virtualenvs/socialhome/bin/circusd config/circus.ini
```

Start Circus. It will automatically start on system boot.

```
sudo service socialhome start
```

For a SystemD config file, see *Other Linuxes or newer Ubuntu using SystemD*.

Done!

That wasn't so hard was it?

Navigate to the domain you chose to install Socialhome on and hopefully you will see a landing page. Signups will be open. Unless you want to keep it that way, after creating your own account, you should close the signups to avoid random people signing up to your instance. See configuration tips at *Running an instance*.

If you didn't configure emails, you cannot complete your user account registration without the email confirmation link. See *Confirming user emails via the shell*.

If you want to set your initially created user as admin, see *Admin user*.

7.2.2 Ubuntu (14.04, Ansible)

See this Ansible role.

7.2.3 Other Linuxes or newer Ubuntu using SystemD

Follow the Ubuntu 14.04 guide, tweaking it to your system. For SystemD, try the following service config (for example saved to `/etc/systemd/system/socialhome.service`):

```
[Unit]
Description=Socialhome Django script
After=syslog.target network.target

[Service]
Environment=DJANGO_SETTINGS_MODULE="config.settings.production"
```

```
Environment=PYTHONPATH="/home/socialhome/socialhome"
Environment=SOCIALHOME_HOME="/home/socialhome"
Environment=RQWORKER_NUM=5
Environment=VIRTUAL_ENV=/home/socialhome/.virtualenvs/socialhome

User=socialhome
Group=socialhome

WorkingDirectory=/home/socialhome/socialhome
ExecStart=/home/socialhome/.virtualenvs/socialhome/bin/circusd /home/socialhome/
↪socialhome/config/circus.ini
Restart=always

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

7.2.4 Other platforms

PR's welcome for guides for more platforms!

7.3 Updating

There are certain steps that should always be done when updating your node to new code. This will be a generic list of what to do when updating which should be near enough the same whatever way the app is installed. Commands might vary depending on your OS below.

If using the *Ubuntu (14.04, Ansible)* there is no need to do anything special. Just re-run the role!

7.3.1 Check the changelog

When updating the code, make sure you check the *Changelog* for any notes about the changes. Sometimes extra manual steps might be required or an update could take a long time due to database migrations.

7.3.2 Change to Socialhome user

Change to user `socialhome` for the rest of the guide.

```
sudo su - socialhome
```

7.3.3 Activate virtualenv

```
workon socialhome
```

7.3.4 Pull in latest code or release

To pull in a release:

```
# Replace release tag with the release, for example "v0.3.1"
git fetch && git checkout <release tag>
```

To pull in master branch head:

```
git pull
```

7.3.5 Install Python dependencies

We use the `pip-tools` command to ensure dependencies are at the correct versions.

```
pip-sync
```

7.3.6 Run migrations

```
python manage.py migrate
```

7.3.7 Install statics

```
npm install
node_modules/.bin/bower install
npm run build
python manage.py collectstatic
```

7.3.8 Restart the app

```
sudo service socialhome restart
```

7.3.9 Done!

Check the application and have fun!

7.4 Running an instance

Some notes on running a production instance.

7.4.1 Django admin

The normal Django admin can be found at `/admin`.

7.4.2 Executing the Django shell

Assuming included installation instructions were used, do the following:

```
sudo su - socialhome
workon socialhome
cd socialhome
python manage.py shell_plus
```

7.4.3 Confirming user emails via the shell

You can manually confirm user emails via the shell by running the following:

```
EmailAddress.objects.filter(email=<email>).update(verified=True)
```

This will allow the user to log in without clicking the confirmation email link.

7.4.4 Admin user

To make a user an admin, log in to the shell and execute the following to set the user as superuser:

```
User.objects.filter(username=<username>).update(is_staff=True, is_superuser=True)
```

7.4.5 Backups

Three places should be backed up from the Socialhome instance to ensure recovery in the event of a disaster.

- The database
- Local settings in `.env` (assuming you are using this way to configure the application)
- The path `socialhome/media/` which contains for example image uploads
- The Redis database (example instructions)

7.4.6 Give your instance some visibility

If you want some public visibility to your instance, consider registering it at some lists that track nodes in "The Federation". Here are a few:

- https://the-federation.info
- https://podupti.me

Why not also contribute to the numbers of the federated social web? Turn on *SOCIALHOME_STATISTICS* to expose some activity counts.

7.4.7 Log files

There are two main logs where Socialhome sends information during runtime.

- Circus process log

  Rotated log files in `/var/log/upstart/socialhome-circus.log`. The location will differ if not using an Upstart based system.

  This log contains the output of all the processes required to run Socialhome, if using the recommended way of running Socialhome using Circus. Any errors for example when starting uWSGI or the worker processes will be found here.

- Application log

  See *SOCIALHOME_LOG_TARGET* configuration value. This log contains logging entries from the application itself. Useful for debugging federation issues or other problems with the actual code.

7.4.8 Configuration

Configuration mainly happens through environment variables. Those are passed to Django via the file `.env` in the repository root. The following items of note can be changed.

After making changes to this file, don't forget to reload the app with `sudo service socialhome restart`.

DATABASE_URL

Default: `postgres:///socialhome`

This must be set to a proper database URL, for example `postgres://socialhome:DATABASEPASSWORDHERE@127.0.0.1:5432/socialhome`.

DJANGO_ACCOUNT_ALLOW_REGISTRATION

Default: `True`

Set this to `False` if you want to disable signups.

DJANGO_ADMIN_MAIL

Default: `info@socialhome.local`

Admin email for example for outgoing emails and providing a feedback channel for users.

DJANGO_ADMIN_NAME

Default: `Socialhome Admin`

Admin display name for example for outgoing emails.

DJANGO_ALLOWED_HOSTS

Default: `socialhome.local`

Domain that is used for this instance. Must be set to the right domain. Note, it's not a good idea to use a sub-domain wildcard for www, ie `.` as per Django docs. Federated sites work better with only one absolute domain.

DJANGO_DEFAULT_FROM_EMAIL

Default: `noreply@socialhome.local`

Set this to the email address that emails should be sent out as.

DJANGO_EMAIL_BACKEND

Default: `django.core.mail.backends.console.EmailBackend`

Must be set to some real email backend if you wish to send emails. See docs for backend options and additional configuration help.

The possible email related additional settings are as follows:

- `DJANGO_EMAIL_HOST` (default `localhost`)
- `DJANGO_EMAIL_PORT` (default `587`)
- `DJANGO_EMAIL_HOST_USER` (default '')
- `DJANGO_EMAIL_HOST_PASSWORD` (default '')
- `DJANGO_EMAIL_USE_TLS` (default `True`)
- `DJANGO_EMAIL_USE_SSL` (default `False`)
- `DJANGO_EMAIL_TIMEOUT` (default '')
- `DJANGO_EMAIL_SSL_KEYFILE` (default '')
- `DJANGO_EMAIL_SSL_CERTFILE` (default '')
- `DJANGO_EMAIL_SUBJECT_PREFIX` (default `[Socialhome]`)
- `DJANGO_SERVER_EMAIL` (default `noreply@socialhome.local`)

Note, email *is* required for signing up. Users will **not** be able to sign up if the instance does not have working email sending.

DJANGO_SECRET_KEY

Default: ''

Must be set to a long secret string. Don't expose it to anyone. See docs

DJANGO_SECURE_CONTENT_TYPE_NOSNIFF

Default: `True`

See docs.

DJANGO_SECURE_FRAME_DENY

Default: `True`

See docs.

DJANGO_SECURE_HSTS_INCLUDE_SUBDOMAINS

Default: `True`

See docs.

DJANGO_SECURE_SSL_REDIRECT

Default: `True`

Redirect all requests to HTTPS. See docs.

REDIS_DB

Default: `0`

REDIS_HOST

Default: `localhost`

REDIS_PASSWORD

Default: ''

REDIS_PORT

Default: `6379`

SOCIALHOME_ADDITIONAL_APPS

Default: `None`

Allows to plug in additional third-party apps, string with comma-separated values, for example `django.contrib.gis,myapp`.

SOCIALHOME_ADDITIONAL_APPS_URLS

Default: `None`

Allows to use additional third-party app url-conf, string with two comma-separated values, url prefix and path to urlpatterns, for example `myapp/,myapp.urls`. If you need to include urls from more than one app, this could be done by creating intermediary app which aggregates urls.

SOCIALHOME_DOMAIN

Default: `socialhome.local`

Must be set to your Socialhome instance domain. Used for example to generate outbound links.

SOCIALHOME_HOME_VIEW

Default: `None`

Allows to use on main page custom view from third-party app, string with path to view, for example `myapp.views.AwesomeHomeView`.

SOCIALHOME_HTTPS

Default: `True`

Force HTTPS. There should be no reason to turn this off.

SOCIALHOME_LOG_TARGET

Default: `file`

Define target for Django and application logs. Possible options:

- `file`, logs will go to a file defined in `SOCIALHOME_LOGFILE`. Note, due to multiple processes logging to the same file, this file log is only really useful for tailing or if running different processes on separate containers or machines.
- `syslog`, logs to syslog, to the `local7` facility.

SOCIALHOME_LOGFILE

Default: `/tmp/socialhome.log`

Where to write the main application log.

SOCIALHOME_NODE_LIST_URL

Default: `https://the-federation.info/socialhome`

URL to make signup link go to in the case that signups are closed.

SOCIALHOME_RELAY_ID

Default: `diaspora://relay@relay.iliketoast.net/profile/`

Which relay instance ID to send outgoing content to. Socialhome automatically integrates with the relay system.

SOCIALHOME_ROOT_PROFILE

Default: ''

If this is set to a local username, that users profile will be shown when navigating to `/` as not logged in user. Logged in users will still see their own profile. Good for single user instances.

SOCIALHOME_STATISTICS

Default: `False`

Controls whether to expose some generic statistics about the node. This includes local user, content and reply counts. User counts include 30 day and 6 month active users.

SOCIALHOME_STREAMS_PRECACHE_SIZE

Default: `100`

Amount of items to store in stream precaches, per user, per stream. Increasing this setting can radically increase Redis memory usage. If you have a lot of users, you might consider decreasing this setting. See *Precaching*.

SOCIALHOME_SYSLOG_FACILITY

Default: `local7`

Define the logging facility for syslog, if `SOCIALHOME_LOG_TARGET` is set to `syslog`.

SOCIALHOME_SYSLOG_LEVEL

Default: `INFO`

Define the logging level of syslog logging, if `SOCIALHOME_LOG_TARGET` is set to `syslog`. Possible options: `DEBUG`, `INFO`, `WARNING`, `ERROR`.

7.5 API

Socialhome has a REST API. This allows to build clients, bots and alternative frontends.

Note, some parts of the API are still work in progress and thus changes could still happen.

7.5.1 API routes

The API methods and data can be browsed using the endpoint `/api/`. Shown endpoints and data depends on your user credentials (log-in using the menu if not already).

7.5.2 Authenticating

The API supports two authentication methods:

Session authentication

This means when you are logged into Socialhome, you automatically have usage of the API from the browser. Note however that POST/PUT/PATCH methods will require CSRF tokens.

Token authentication

API authentication can happen by setting the required HTTP header as follows:

```
Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b
```

Your client can obtain a token for the user by posting `username` and `password` as form data or JSON to the view `/api-token-auth/`. This will return a token as follows:

```
{ 'token' : '9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b' }
```

Users can also retrieve and regenerate tokens from the UI from their profile menu.

7.5.3 Development

The API is made with Django REST Framework. Help is welcome to expand the API!

7.5.4 Clients

See the *Clients* section.

7.6 Clients

This page will have a list of clients using the Socialhome *API*. Please send PR's if you have made one!

7.6.1 shcli

This is the official Python library and command line client.

See documentation and code on GitHub.

7.7 Community

7.7.1 Official project account

To keep up to date with the project you can follow the official project account from Socialhome or compatible software (Diaspora, Friendica, Hubzilla, GangGo). The official account is `hq@socialhome.network` (https://socialhome.network/u/hq/).

7.7.2 Feedback and community chat

We have a few chat channels you can join. All of these are bridged so you only need to join with which ever is your favourite to use.

- FreeNode IRC, channel `#socialhome` (webchat here)
- Gitter chat
- Matrix room `#socialhome:matrix.org`

Join in if you have questions regarding installation, development or usage, or just to say hi!

7.8 Development

Socialhome is missing features and needs a lot of polish on the UI side. If you are familiar with Django (or want to learn!) and are interested in getting involved, please don't hesitate to get in touch!

For guidelines how to contribute, please first read the *Contributing* guide.

- Source code repo
- Issue tracker
- Kanban board

7.8.1 Environment setup

Instructions are for Ubuntu 16.04+ (+ simple Alpine 3.6 dependencies script). Please contribute via PR's if you notice anything missing or want to contribute instructions for another platform.

Python Virtualenv

Python 3.4, 3.5 and 3.6 are officially tested against. Ensure the following are installed:

- Python system dependencies
- NodeJS (version 6+)
- PostgreSQL server
- Redis

The file `requirements.apt` contains other various dependencies. You can use the `install_ubuntu_dependencies.sh` script to help installing these.

You can use the `install_alpine_dependencies.sh` script to install required dependencies (including Python, NodeJS, PostgreSQL and Redis) on Alpine.

Install Python dependencies

We use `pip-tools` as the way to install Python dependencies. All the "base" dependencies, including production deployment dependencies are locked in `requirements.txt`. The file `dev-requirements.txt` includes both the base and the extra development/testing related dependencies.

To use `pip-tools`, first install it:

```
# Ensure pip and setuptools are up to date as well
# We need a slightly older setuptools due to a bug in pip-tools
pip install -U pip setuptools==30.4 pip-tools
```

Then install dependencies:

```
# Production environment
pip-sync

# Development environment
pip-sync dev-requirements.txt
```

It is not mandatory to use `pip-tools` for running a production installation. For development it is mandatory. All dependencies should be placed (unlocked) in either `requirements/requirements.in` (base) or `requirements/requirements-dev.in` (development extras). Then execute `./compile-requirements.sh` to update the locked dependency files after each change to the `.in` files. See pip-tools for more information.

Do NPM, Bower

```
npm install
bower install
sudo npm -g install grunt
npm run dev
```

To watch files and build bundles automatically, use this.

```
npm run watch
```

Configure

Configuration is done via environment variables. For the meaning of them, look them up under files in `config/settings`. Values in the file `.env` will be used automatically.

```
cp .env.example .env
```

Edit any values necessary. By default the `SECRET_KEY` is empty. You MUST set something to it. We don't supply a default to force you to make it unique in your production app.

Create a database

If you changed the `DATABASE_URL` in the settings file, make sure to change the values in these commands accordingly.

```
sudo su - postgres
createuser -s -P socialhome  # give password 'socialhome'
createdb -O socialhome socialhome
exit
python manage.py migrate
```

Running the development server

Just use the standard command:

```
python manage.py runserver
```

Unfortunately `runserver_plus` cannot be used as it does not integrate with Django Channels.

Creating a user

To create an *superuser account*, use this command:

```
python manage.py createsuperuser
```

After this you need to log in once with the user via the user interface (which creates an email confirmation) and then run the following in the Django shell to confirm the email:

```
EmailAddress.objects.all().update(verified=True)
```

You should now be able to log in as the user `admin`.

Search index

The search indexes must be initialized, otherwise there will be an error when trying to use search. Run this command once:

```
python manage.py rebuild_index
```

Any further changes to indexes objects will be maintained automatically from this point onwards. If you ever need to rebuild the index from scratch, use the same command.

7.8.2 Running tests

The user needs right to create databases:

```
# On some distributions, regular users are not sudoers; you may need to type:
#    su -
su - postgres
psql -c "ALTER USER socialhome CREATEDB;"
```

Python tests

```
py.test
```

JavaScript tests

Legacy frontend

This will launch a separate `runserver` on port 8181 and execute the tests against that. The separate `runserver` instance will be killed after the tests have been executed.

```
grunt test
```

New Vue based frontend

Execute the following to run the new frontend JavaScript tests.

```
npm run test
```

7.8.3 API Routes

There is a dependency in the API route URL configurations with the new Vue based frontend tests. If you change or add new API routes during development, you must also do the following:

```
python manage.py collectstatic_js_reverse
mv staticfiles/django_js_reverse/js/reverse.js socialhome/streams/app/tests/fixtures/
↪Url.js
```

This updates the JavaScript fixtures with the new URL configuration.

7.8.4 Linters

ESLint

There is an `.eslintrc` provided. We follow the Airbnb and Vue guidelines with some tweaks. It's recommended to add this configuration to your editor directly. To run ESLint directly, use the following command. NOTE! This is only valid for the new Vue based frontend, not JS in `socialhome/static`.

```
npm run lint
```

7.8.5 Building local documentation

```
cd docs
make html
```

7.8.6 Doing a release

Bump version number in three places:

- `socialhome/__init__.py`
- `docs/conf.py`
- `docs/changelog.rst`

To generate a markdown version of the release changelog, first install `Pandoc`:

```
sudo apt install pandoc
```

Then execute the following and copy the markdown version for pasting to GitHub releases or a Socialhome post:

```
pandoc --from rst --to markdown_github docs/changelog.rst | less
```

After the release commit has been pushed and a release has been tagged, set a development version in the same above files. This is basically the next minor release postfixed by `-dev`.

Commit and author stats

Some commands to get nice stats for release posts.

Authors

```
git shortlog -s -n -e <first release commit>..HEAD --no-merges
```

Changes

```
git diff --stat <first release commit>..HEAD
```

7.8.7 Developing with Docker

If you choose, you may develop Socialhome using Docker, rather than installing Postgres and Redis manually on your computer.

Supported versions

This guide assumes you are running Docker on a GNU/Linux based system such as Ubuntu, Debian or Fedora Linux. It may be possible to run this on other platforms where Docker is supported, but those are untested.

The docker development installation was tested on Docker version 17.09 and docker-compose 1.16.1.

Steps

The first step is to copy the example docker-compose file `docker/dev/docker-compose.yml.example` file to the root of the project. eg

`cp docker/dev/docker-compose.yml.example ./docker-compose.yml`

You also need to set an .env file as per the above instructions. Use the `.env.example` as a starting point.

From there, you can build the images:

`docker-compose build`

And then the steps you would normally do, but throught he django image, ala:

`docker-compose run django manage migrate` and `docker-compose run django manage createsuperuser`

And then just

`docker-compose up`

Defaults

The defaults are that that the Docker image will be running on port 8000 and then exposed to the host OS on the same port (ie you can browse to http;//localhost:8000 to see the Django instance running). Redis and Postgres will be running but not exposed to the host OS by default. These can be changed on the `docker-compose.yml` file.

7.8.8 Generating dummy content

There is a management command to generate a bunch of dummy `Content` objects. Please feel free to expand it with more configuration options and different types of content. To use it, run the following:

```
python manage.py create_dummy_content
```

`--help` will give you available options.

7.8.9 Contact for help

See our communication channels in the *Community* page.

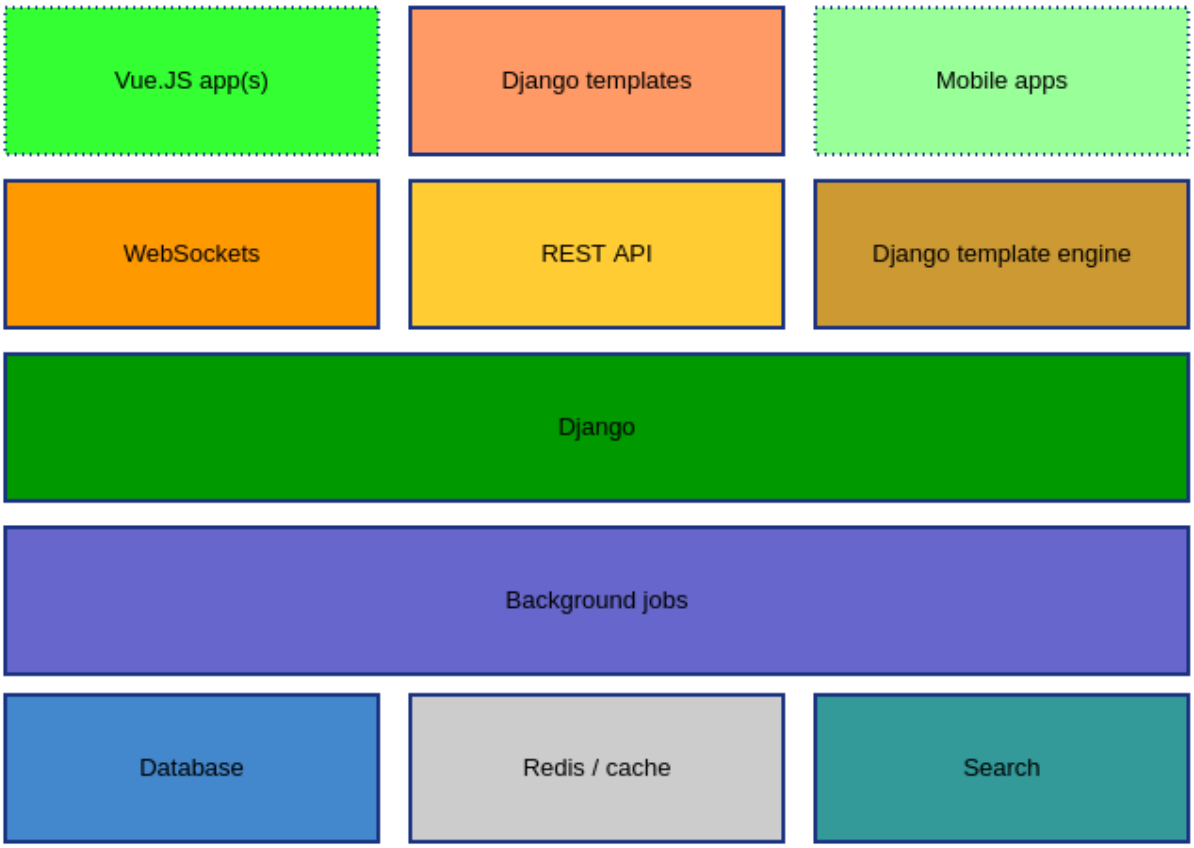
You can also ask questions or give feedback via issues.

7.9 Architecture

Some details on the architecture and future high level plans for Socialhome.

7.9.1 Component map

Our current and future (dotted lines) components look something like this:



At the lowest level we have the database (**PostgreSQL**) and **Redis** based cache / queue storage. Search is currently powered by **Django-Haystack + Whoosh**.

On top of this we have the background jobs, which are powered by **RQ**.

In the middle sits **Django**.

To provide data for the frontend we have 3 solutions - **WebSockets** (powered by **Channels**), a **REST API** powered by **Django REST framework** and **Django** template engine itself.

For the frontend, we will have 3 solutions. Currently everything is **Django templates**. We will want to keep some of the pages as Django templates. For the streams (and possibly other pages), we want to create a **Vue.js** app. Additionally, **mobile apps** would be provided.

7.9.2 Component and feature notes

Vue.js app(s)

The current streams JavaScript is largely based on lots of **jQuery** events modifying the DOM. Since the streams can grow to be quite big, this results in very bad performance. Additionally, the code is beginning to get hard to read and difficult to modify without regressions ("spaghetti code").

What we want to do is rewrite the streams as a more modern performant JS application. For the framework, discussion has been centering around **Vue.js**. The rationale is that Vue has the benefits of React.js with less overhead in learning curve and development time.

Why not replace the whole frontend with Vue.js? Simply because we want to use the best job for each software area. Some of the pages will not benefit from being rewritten in JavaScript. Django templates are powerful and fast to develop. But some parts, like the streams, will benefit hugely from features like the virtual DOM provided by Vue, in addition to allowing cleaner JavaScript code base.

Code layout

Socialhome code layout is split into logical Django apps based on the feature provided. The JS code should follow this pattern and live in the respective app. For example for the `streams` Vue.js application, the following code layout would make sense:

```
socialhome/
  streams/
    app/
      components/
        (components)
      App.vue
      main.js
    templates/
      streams/
        app.html
    views.py
```

Basically the idea is that `views.py` contains a Django view that loads the template inheriting from `base.html`. The template then injects the Vue app, loading the stream. To speed up rendering we provide some initial stream data in the Django template context, then continuing everything via the REST API.

All the Vue apps build configuration should be on the top level of Socialhome, set up so all the apps build using the same `npm` commands. Each Vue.js app should however generate its own JS bundle file.

Code style

For the new Vue.js based JavaScript we should follow the popular Airbnb guidelines with the following exceptions:

- No semicolons. This is a Python project, we can go for more Pythonic looking JavaScript.

All code should be allowed ES7 features, using Babel to transpile.

Tests

We should use standard testing tools for the Vue apps code, for example **Karma** + **Mocha**.

Timeline

Since this is a huge task which cannot be done at once, the new Vue.js based streams will be provided in addition to the current streams served by Django templates. This could be done in phases:

1. Alpha, little functionality - Render using Vue.js if a parameter *?vue* passed in the url.
2. Beta, most of the functionality present - Allow user to go to preferences and choose whether to see the new or legacy stream.
3. Final, all functionality covered - Make Vue based streams default, removing the old streams code.

Tracking issue

The Vue.js streams rewrite is tracked in this issue.

Search

Search is currently powered by django-haystack as the framework and Whoosh as the engine. Whoosh is a pure Python backend with a file based search index. As performance requirements increase (for example full text content search), we should offer the option to use Elasticsearch as an optional search backend. Django-haystack supports both backends with just configuration changes. Whoosh should still be the default since it doesn't require extra installations like Elasticsearch does.

The global search works as follows, in this order:

- Search by profile handle:
  - If direct match found -> render profile
  - If remote match found -> fetch and render profile
- Search all indexes for any matches

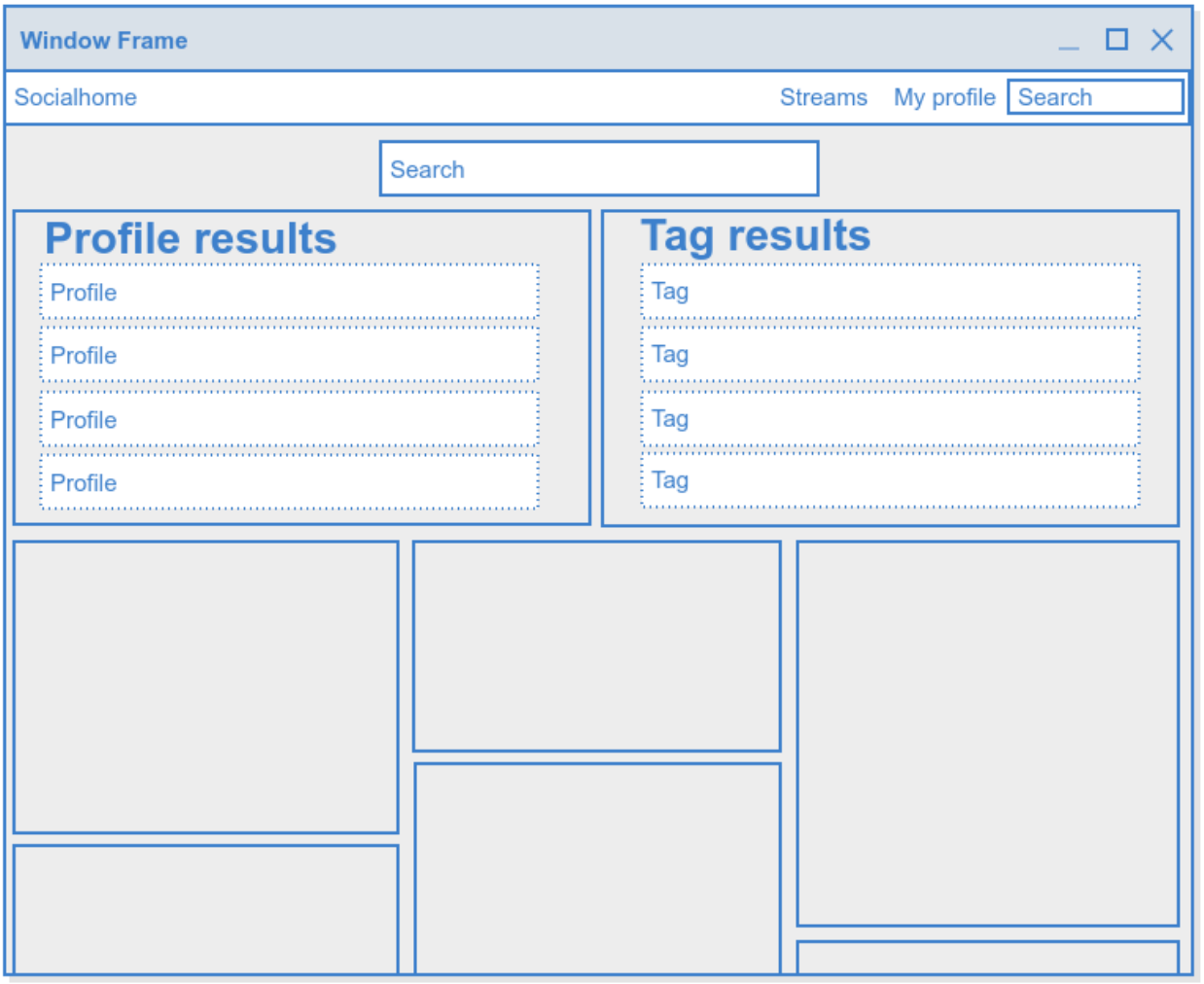
Indexes

Currently a search index only exists only for profile objects. The plan is to add the following search indexes:

- Tags
- Content

UI

Profiles and tags can be easily listed in a list or table structure. Content would make sense to be rendered in a normal grid. This would make the search results page just another (dynamic) stream. See below mockup.



Streams

There are many streams in Socialhome. The main streams are user profiles, followed and the public stream, but basically each single content view is also a stream. Opening a reply in an individual window would also create a stream for that reply content. Additionally, we want users to be able to create custom streams according to rules. For example, a stream could be "followed profiles + tag #foobar + tag #barfoo".

A stream should automatically subscribe the user using websockets and handle any incoming messages from the server (currently in `socialhome/stream/app/main.js`), notifying the user of new content and adding it to the page on request (without a page load).

This basic design should be kept in mind when touching stream related code.

Stream templates

Note: This section relates to the old Django templates + jQuery stream. For the Vue.js streams, see above.

Content in streams in is visualized mainly as content grid boxes. This includes replies too, which mainly use the same template code.

There are a few locations to modify when changing how content is rendered in streams or the content detail view:

- `socialhome/streams/templates/streams/base.html` - This renders the initial stream as a basic Django template on page load.
- `socialhome/streams/templates/streams/_grid_item.html` - Renders actual content item in initial stream and content detial.
- `socialhome/static/js/content.js` - This is the main JavaScript template which is used to insert content into the stream. This is used for both top level content and replies in content streams.

All these templates must be checked when any content rendering related tweaks are done. Note however that actual content Markdown rendering happens at save time, not in the templates.

Precaching

To make complex streams load fast, we precache them in Redis. The precache streams are updated on content save time.

Each stream has an Ordered Set for each user with the following data:

```
key = sh:streams:<stream_name>:<user_id>
score = <time>
value = <content.id>
```

Additionally, each stream has a Hash for each user with the "through ID's". A through ID is the content which caused the cached content to be added into the stream. Normally this would be the cached content itself, but for shares, this would be the share content ID. The Hash is as follows:

```
key = sh:streams:<stream_name>:<user_id>:throughs
field = <content.id>
value = <through content.id>
```

Only expensive streams are precached. This includes any stream which will pull up shares (for example "Followed" and "My content (all)"). Additinally any custom streams should always be precached for fast reads. An example of a stream which is not precached is the "Public" stream.

7.10 Contributing

Want to contribute to Socialhome? Great! <3 Please read on for some guidelines.

7.10.1 First things first

Please make sure you have some knowledge of what the software is for before jumping in to write code. You don't even have to install a development environment. Just head to https://socialhome.network, create an account and play around.

If you already are a user or run your own instance, you probably have some ideas on how to contribute already. The best contributions come from real personal need.

7.10.2 Finding things to do

We have an issue tracker on GitHub. If you don't already have an idea on what to do, check out the issues listed there. Some issues are labeled as newcomer. These are easy picking tasks for those either new to Socialhome or with less knowledge of Django.

See also our *Architecture* for high level plans.

7.10.3 Logging issues

Contributions are not just code. Please feel free to log not only bugs but also enhancement ideas in the issue tracker. For issues that have not been confirmed (= they don't have the label "ready"), triaging is important contribution also. Reproducing and supplying more information on these issues is valuable contributon to the project.

Welp I found a security issue, shall I just file an issue?

7.10.4 Writing code

So you're ready to write code! Great! Please remember though that the project already has a vision, the software has architecture and the project maintainer will have strong opinions on how things should be implemented. Before you write a lot of code that even remotely feels like it would need a design decision, please *always* discuss your plan first with the project maintainer. Otherwise you might spend a lot of time writing code only to be told the code will not be merged because it doesn't fit into the grand plan.

Please don't be afraid to get in touch, see channels in the *Community* pages.

7.10.5 Creating pull requests

Before submitting a pull request, please ensure you've read and understood the following checklist.

Checklist

Please review the guidelines for contributing (https://socialhome.readthedocs.io/en/latest/contributing.html) to this repository and then go through the following checklist.

- Make sure you are requesting to merge a non-master branch from your fork. Do not create PR's from your *master* branch!
- Does your code have unit tests? All major code paths should be tested sufficiently. See existing tests for examples.
- Did you run the whole test suite through and ensure no existing tests break?
- Please check the documentation whether your PR will require changes or additions to any documentation pages. Use proper English!
- If changing or adding UI pages or elements add screenshots to the PR.
- If the PR is not ready to be merged:
  - Prefix the PR title with "[WIP] ".
  - Add a list of TODO's that are missing from the PR to the description.
- Consider adding a changelog entry to the current unreleased version. Use proper English, reference the right issue (if any, not your PR) and make sure to add the entry to the correct section ("Added", "Changed", "Fixed" or "Removed"). If you are not sure whether you need a changelog entry - don't add one!

- Ensure commit messages are descriptive and sufficiently detailed. If possible, split the feature into smaller easier to follow commits. As a rule of thumb, small PR's and small commits are always preferred. See https://chris.beams.io/posts/git-commit/ for notes on what is a good commit.

Thank you!

7.10.6 Reviewing code

Code review is a valuable way to contribute, and also to learn about the code base! Don't be afraid to give some comments to open pull requests! You don't have to be a veteran or know everything to be able to give opinions. Pull request reviews are not just for reviewing, they're a valuable opportunity for learning too.

7.10.7 Tests

As a general rule all code must have unit tests. For bug fixes provide a test that ensures the bug will not be back and for features always add a good enough coverage. PR's without sufficient test coverage will not be merged.

Testing tools

Django

We use `py.test` as test runner but the tests themselves are Django based test classes. We have our own base classes which should be used as a base for all Django tests. Some old tests are pure py.test function based tests, feel free to convert these to Django test classes.

Focus is placed in pure unit tests instead of complex integration or browser tests. In terms of coverage, 100% is not the key, meaningful tests and coverage of critical lines is. Don't worry if a PR drops coverage a bit if the coverage diff clearly shows all critical code paths are covered by meaningful tests.

Vue

The JS tests are using the Avoriaz Vue test utils and Mocha test runner.

7.10.8 Code style

Python

As a general rule, for Python code follow PEP8, except with a 120 character line length. We provide an `.editorconfig` in the repository root.

JavaScript

There is an `.eslintrc` configuration provided.

Alphabetical ordering

When possible, try to always make all list items, dict keys, class methods, classes / functions in file, etc, everything alphabetically organized. This helps finding things when files grow and classes get a lot of methods. Sometimes this is not possible, for example when classes subclass another class in the same file. In this case for example, alphabetical ordering can be forgotten for logical placement.

7.10.9 Python dependencies

We use `pip-tools` as the way to install Python dependencies. All the "base" dependencies, including production deployment dependencies are locked in `requirements.txt`. The file `dev-requirements.txt` includes both the base and the extra development/testing related dependencies.

To use `pip-tools`, first install it:

```
# Ensure pip and setuptools are up to date as well
# We need a slightly older setuptools due to a bug in pip-tools
pip install -U pip setuptools==30.4 pip-tools
```

Then install dependencies:

```
# Production environment
pip-sync

# Development environment
pip-sync dev-requirements.txt
```

It is not mandatory to use `pip-tools` for running a production installation. For development it is mandatory. All dependencies should be placed (unlocked) in either `requirements/requirements.in` (base) or `requirements/requirements-dev.in` (development extras). Then execute `./compile-requirements.sh` to update the locked dependency files after each change to the `.in` files. See pip-tools for more information.

7.11 Brand

Documentation relating to branding of Socialhome as a product. Each server in the network is free to apply whatever branding for their particular server.

When talking about Socialhome as software or a platform, the following graphics and colour schemes should be used.

7.11.1 Logo

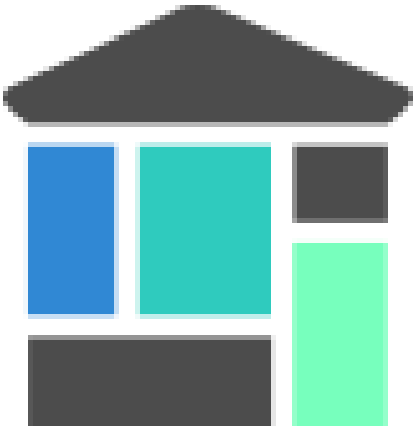
Our logo is available as an SVG and PNG's of various sizes. The logo comes in a dark and light variant, for different backgrounds.

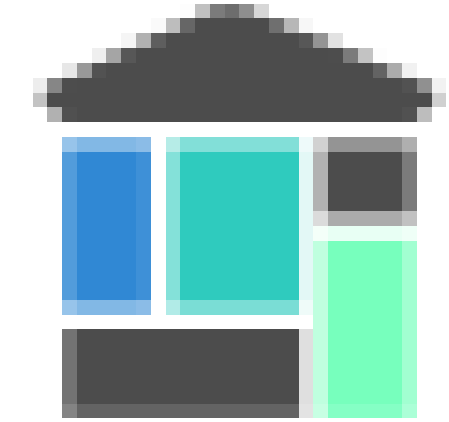
The logo is contributed by lostinlight, licensed under WTFPL.

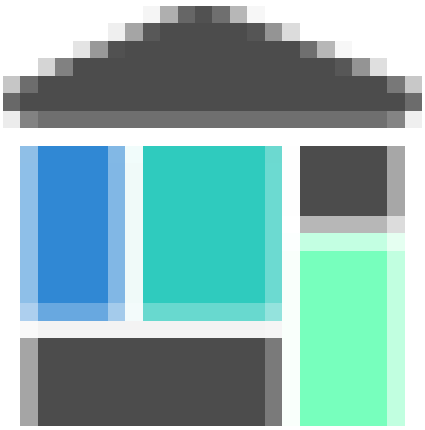
Dark

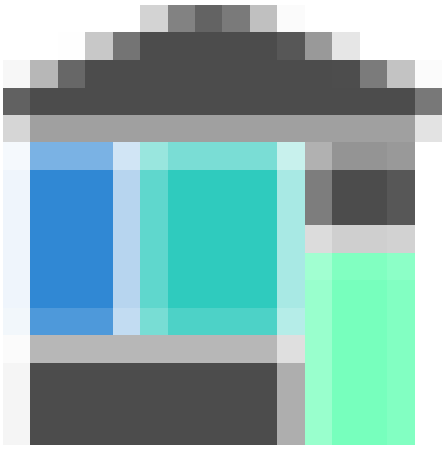
SVG

PNG





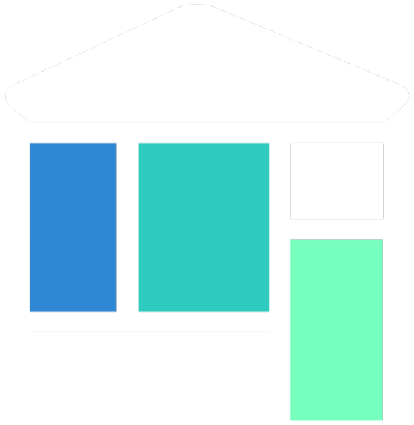


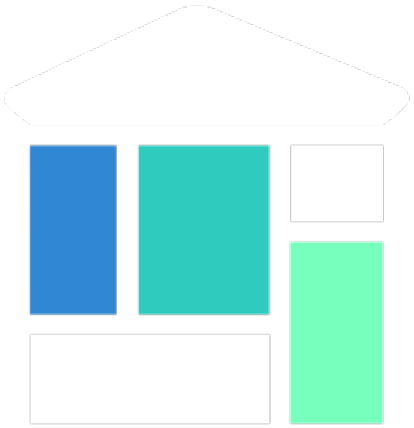


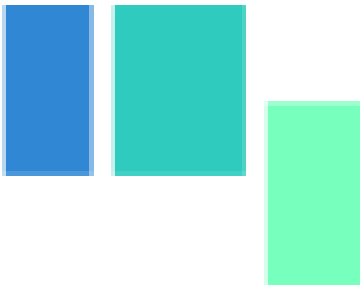
Light

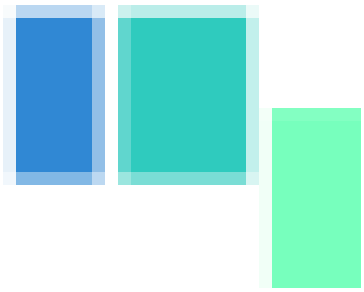
SVG

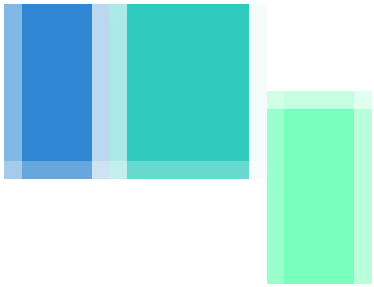
PNG

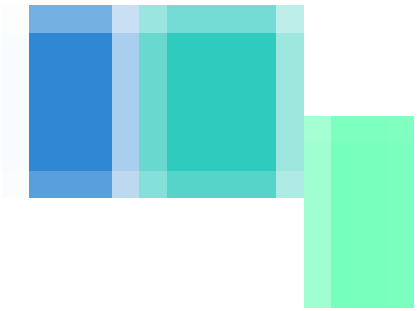












7.11.2 Stickers

Feel free to print these out and spread the love!

The stickers are contributed by lostinlight, licensed under CC-BY-4.0.

PDF's available here.

7.12 FAQ

Generic questions regarding Socialhome.

7.12.1 If I run an instance, will it automatically connect to the network?

No, Socialhome doesn't work in a peer2peer sense. Incoming content does not happen automatically, outbound does in a limited sense. Federation happens on two main principles:

- Social relationships

  This means content you create will be sent to the servers that your followers are on. Content that users you follow will send content they create to your server. The more inter-server relationships users on the server you

are on, the more content will reach your server. This is the standard federation model in use by the Diaspora, OStatus and ActivityPub powered networks.

- The public content relay system

  The relay system is a network of servers that exist for one single purpose, to receive public content and to distribute it to places it would not otherwise reach (with social relationships based federation). Socialhome by default integrates with the relay system by sending all public content to it and subscribing to all public content on the relays. To receive content, this requires registering the Socialhome server to the relays so that they know about the server and can start sending content.

  Registration happens at The-Federation.info. For more technical details about the relay system, check the reference server documentation.

As a recap:

- To receive content from around the network, a new server can do the following:
  - Following other users on other servers to create social relationships. This happens by using the full handle (for example `hq@socialhome.network`) of the user.
  - Registering to the relay system.
- To send out content to the network, a new server can do the following:
  - Following other users on other servers to create social relationships. Public content will also be sent out to the servers whose users *you* follow, not just who follow you.
  - No need to register with the relay. All public content created on a Socialhome server is by default sent out to the relay system.

7.12.2 Welp I found a security issue, shall I just file an issue?

No, please report security issues directly to the maintainers in private messages. Reporting security issues publicly in the issue tracker would allow other people to use the security issue to their benefit before a fix is released. Reporting security issues to the maintainers in private allows us to fix the issue and roll out a fix hopefully before many users are affected.

Please report security issues directly to the maintainer by email at `mail@jasonrobinson.me`.

7.13 Changelog

7.13.1 0.8.0-dev (unreleased)

Added

- RFC3033 webfinger support for Diaspora protocol (#405)

  This allows better profile discovery by remote non-Socialhome servers.

Changed

- Setting `SOCIALHOME_RELAY_DOMAIN` is now called `SOCIALHOME_RELAY_ID`. We're slowly replacing all direct Diaspora handle references with Diaspora URI format profile ID's in preparation for ActivityPub protocol addition.

No action needed from server admins unless you have changed this setting, in which case it should be updated accordingly.

- Start sending profile changes to remote nodes as public messages for better efficiency

Fixed

- Fix precalculated streams maintenance job.

  Due to mistake in regexp not all old precalculated stream items were pruned in maintenance. Now fixed which should ensure Redis memory usage does not suffer from unreasonable increase over time.

7.13.2 0.7.0 (2018-02-04)

New Vue.js frontend

The work that started at a small hackathon in Helsinki in July 2017 is finally finished! The old buggy and hard to maintain Django template + jQuery based frontend has been completely rewritten in Vue.js. This provides a modern frontend code base, making it possible to add new features faster and to spend less time fixing bugs in the spaghetti code.

A huge thanks goes out to @christophehenry doing most of the work in pushing this rewrite through!

Added

- Possibility to skip adding an OEmbed or OpenGraph preview to content. (#364)

  There is a new checkbox on content create that allows skipping adding a link preview to the content.

- Add maintenance job to groom precache information from Redis. This ensures Redis memory usage stays stable.

  Important for server admins. There is a new process to run that is responsible for scheduling these maintenance jobs. The

  - If you already use the provided Circus configuration to run Socialhome, you **don't need to do anything**. When you restart Socialhome, the updated Circus configuration will automatically be used and the scheduler process started by Circus.
  - If you have a custom setup, preferring to run all processes manually, ensure one `rqscheduler` process is running at all times to ensure maintenance jobs and other future scheduled jobs are executed.

  A new configuration item `SOCIALHOME_STREAMS_PRECACHE_SIZE` is available to set the maximum size of precached stream items per user, per stream. This defaults to 100 items. Increasing this setting can radically increase Redis memory usage. If you have a lot of users, you might consider decreasing this setting if Redis memory usage climbs up too high.

- It is now possible to use email for log-in. (#377)
- Added a Code of Conduct document. All contributors to Socialhome are expected to honour these simple rules to ensure our project is a safe place to contribute to.

  Read the Code of Conduct here.

- Profile API has 4 new read only fields:
  - `followers_count` - Count of followers the given Profile has. For remote profiles this will contain only the count of followers on this server, not all the followers the profile has.

  - `following_count` - Count of local and remote profiles this Profile is following. For remote profiles this will contain only the count of profiles following this profile on this particular server.
  - `has_pinned_content` - Boolean indication whether the local profile has pinned any Content to their profile stream. Always false for remote profiles.
  - `user_following` - Boolean whether logged in user is following the profile.
- There is now a management command to generate dummy content for development environment purposes. See *Development* pages.
- Installation docs now have an example SystemD service configuration, see *Other Linuxes or newer Ubuntu using SystemD*. (#397)
- Content API has a new read only field `has_twitter_oembed`. This is `true` if the content text had a Tweet URL *and* a fetch for the OEmbed code has been successfully made.
- Content create page now has an option to disable federating to remote servers when saving the content. (#296)

  The content will still update to local streams normally. Federating the content can be enabled on further saves.
- If signups are closed, the signup link will now stay active but will point to a list of Socialhome nodes. (#354)

  By default this URL is https://the-federation.info/socialhome, but can be configured by the server admin.

Changed

- When processing a remote share of local content, deliver it also to all participants in the original shared content and also to all personal followers. (#206)
- Allow creating replies via the Content API.

  Replies are created by simply passing in a `parent` with the ID value of the target Content. It is not possible to change the `parent` value for an existing reply or root level Content object once created. When creating a reply, you can omit `visibility` from the sent data. Visibility will be used from the parent Content item automatically.
- Removed Opbeat integration related configuration. The service is being ramped down. (#393)

  If as a server administrator you have enabled Opbeat monitoring, it will stop working on this update.
- New VueJS stream is now default o/ (#202)

  Old stream can still be accessed using the user preferences or by passing a *vue=0* parameter in the URL. All existing users have been migrated to use the new VueJS streams by default.

Fixed

- Redirect back to profile instead of home view after organize pinned content save action. (#313)
- Fix searching of an unknown remote profile by handle using uppercase letters resulting in an invalid local profile creation.
- Fix Content querysets not correctly including the 'through' information which tells what content caused a share to be added to a stream. (#412)

  This information was already correctly added in the streams precalculation phase, but if the cache started cold or a viewing user cycled through all cached content ID's and wanted some more, the database queries did not return the right results.

- Attempt to fetch OEmbed and OpenGraph previews of URL's in content in the order of the links found. (#365)

  Previous behaviour lead to fetching previews of urls in random order, leading to a different url preview on different Socialhome servers.

- Fix remote profile retrieval from remote servers which don't support legacy Diaspora protocol webfinger. (#405)

  New version of federation library defaults to trying the new style webfinger with a fall back to legacy.

7.13.3 0.6.0 (2017-11-13)

Added

- Profile "All content" streams now include the shares the profile has done. (#206)
- Streams API now has endpoints for profile streams to match the profile streams in the UI. (#194)
  - `/api/streams/profile-all/{id}/` - fetches all content by the given profile (including shares), ordered by created date in reverse order (= new stuff first).
  - `/api/streams/profile-pinned/{id}/` - fetches pinned content by the given profile, ordered as set by the profile owner.
- New fields added to Content API:
  - `is_nsfw`, boolean value, `true` if the content text has the tag `#nsfw` in it.
  - `share_of`, if the `content_type` is `share`, this will contain the ID of the shared Content.
- If an incoming share references a remote target that doesn't yet exist locally, it and the author profile will be fetched and imported over the network. (#206)
- There are now Docker files for doing development work for Socialhome. See the docs here.
- Third-party applications can now be added to enhance Socialhome or replace some of the core functionality, using configuration. The following new settings are available:
  - `SOCIALHOME_ADDITIONAL_APPS` - List of additional applications to use in Django settings.
  - `SOCIALHOME_ADDITIONAL_APPS_URLS` - Additional third-party URL's to add to core url configuration.
  - `SOCIALHOME_HOME_VIEW` - Override the home view with another view defined with this setting.
- Content API now has a new `shares` endpoint. (#206)

  This allows retrieving all the shares done on a Content.

- We now have a logo

The logo also comes in a light version, for dark backgrounds. See *Brand* for details.

Changed

- Logging configuration changes:
  - Removed separate logfile for the federation loggers. Now all logs go to one place. Setting `SOCIALHOME_LOGFILE_FEDERATION` has been removed.
  - Added possibility to direct Django and application logs using a defined level to syslog. Adds three settings, `SOCIALHOME_LOG_TARGET` to define whether to log to file or syslog, `SOCIALHOME_SYSLOG_LEVEL` to define the level of syslog logging and `SOCIALHOME_SYSLOG_FACILITY` to define the syslog logging facility. See configuration documentation.
- **Important!** The file to place configuration environment variables has changed to `.env`.

  This is a more standard file name for environment variables than the previous `env.local`. For now we'll still load from the old file too, but a warning will be displayed to rename the file.
- **Breaking change**. API `Content` serialization now returns list of tags as *name of tag*, not ID as before. The names do not contain the character "#".
- Content API `replies` endpoint now includes all the replies on the shares of the Content too.
- Use modified timestamp for created timestamp when federating out to remote nodes. (#314)

  This makes edits federate more reliably to some remote platforms that support edits.
- Stream grid item reply icon changed from "envelope" to "comments". (#339)

Fixed

- Fix various issues with OpenGraph tags parsing by switching to self-maintained fork of `python-opengraph`.
- Share button is no longer visible if not signed in (#325)
- Remote profile image urls that are relative are now fixed to be absolute when importing the profile from remote (#327)
- Fix poor performance of fetching replies.

When adding replies of shares to the collection of replies fetched when clicking the reply icon in the UI, a serious performance regression was also added. Database queries have now been optimized to fetch replies faster again.

- When editing a reply, the user is now redirected back to the parent content detail view instead of going to the reply detail view. (#315)
- Fix regression on visibility of remote replies on shares.

  Replies inherit the parent object visibility and share visibility defaults to non-public in the federation library. Diaspora protocol removed the `public` property from shares in a recent release, which meant that we started getting all shares as non-public from the federation layer. This meant that all comments on the shares were processed as non-public too.

  With a change in the federation layer, Diaspora protocol shares are now public by default.
- Fixed Streams API content `user_is_author` value always having `false` value.

7.13.4 0.5.0 (2017-10-01)

Python dependencies

Switched to `pip-tools` as the recommended way to install Python dependencies and cleaned the requirements files a bit. Now all the "base" dependencies, including production deployment dependencies are locked in `requirements.txt`. The new file `dev-requirements.txt` includes both the base and the extra development/testing related dependencies.

To use `pip-tools`, first install it:

```
pip install -U pip-tools
```

Then install dependencies:

```
# Production environment
pip-sync

# Development environment
pip-sync dev-requirements.txt
```

It is not mandatory to use `pip-tools` for running a production installation. For development it is mandatory. All dependencies should be placed (unlocked) in either `requirements/requirements.in` (base) or `requirements/requirements-dev.in` (development extras). Then execute `./compile-requirements.sh` to update the locked dependency files after each change to the `.in` files. See pip-tools for more information.

Added

- GIF uploads are now possible when creating content or replies. (#125)
- Content API has a new endpoint `/api/content/<id>/replies/`. This returns all the replies for the given content.
- Shares made by followed contacts are now pulled up to the "Followed" stream.

  This happens only if the user has not already seen this content in their "Followed" stream. Each content should only appear once, either directly by following the author or a followed contact sharing the content. Multiple shares do not raise the content in the stream again.

Changed

- Rendered link processing has been rewritten. This fixes issues with some links not being linkified when rendering. Additionally now all external links are made to open in a new tab or window. (#197)
- Previously previews and oEmbed's for content used to only pick up "orphan" links from the content text. This meant that if there was a Markdown or HTML link, there would be no link preview or oEmbed fetched. This has now been changed. All links found in the content will be considered for preview and oEmbed. The first link to return a preview or oEmbed will be used.
- Streams URL changes:
  - All streams will now be under `/streams/` for a cleaner URL layout. So for example `/public/` is now `/streams/public/`.
  - Tag stream URL has been changed from `/streams/tags/<tag>/` to `/streams/tag/<tag>/`. This small change allows us to later map `/stream/tags/` to the tags the user is following.

  Since lots of old content will point to the old URL's, there will be support for the legacy URL's until they are needed for something else in the future.
- **Breaking change**. Profile API field changes:
  - Added:
    - `url` (Full URL of local profile)
    - `home_url` (Full URL of remote profile, if remote user)
    - `is_local` (Boolean, is user local)
    - `visibility` (Profile visibility setting, either `public`, `limited`, `site` or `self`. Editable to self)
  - Removed (internal attributes unnecessary for frontend rendering):
    - `user`
    - `rsa_public_key`
- **Breaking change**. Content API field changes:
  - Added:
    - `timestamp` (ISO 8601 formatted timestamp of last save)
    - `humanized_timestamp` (For example "2 hours ago")
    - `url` (Full URL to content detail)
    - `edited` (Boolean whether content has been edited since creation)
    - `user_following_author` (Boolean whether current user is following content author)
    - `user_is_author` (Boolean whether current user is the author of the content)
    - `user_has_shared` (Boolean whether current user has shared the content)
  - Changed:
    - `author` is now a limited serialization of the author profile, containing the following keys: "guid", "handle", "home_url", "id", "image_url_small", "is_local", "name", "url".

      The reason for serializing the author information to content is related to privacy controls. A user who maintains a limited profile can still create public content, for example. A user who is able to view the content created by the user should also see some limited information about the creating profile. To get the full profile, the user needs to fetch the profile object by ID, which is subject to the visibility set by the profile owner.

  - Removed (internal attributes unnecessary for frontend rendering):
    * `created`
    * `modified`
    * `oembed`
    * `opengraph`
- Refactoring for streams views to use new Stream classes which support pre-caching of content ID's. No visible changes to user experience except a faster "Followed users" stream.

  A stream class that is set as cached will store into Redis a list of content ID's for each user who would normally see that content in the stream. This allows pulling content out of the database very fast. If the stream is not cached or does not have cached content ID's, normal database lookups will be used.

  This refactoring enables creating more complex streams which require heavier calculations to decide whether a content item should be in a stream or not.

Fixed

- Cycling browser tabs with CTRL-TAB when focused on the editor no longer inserts a TAB character in the editor.
- Don't federate shares to shared content local author. This caused unnecessary deliveries between the same host.

7.13.5 0.4.0 (2017-08-31)

Update notes

This release contains long running migrations. Please allow up to 10 minutes for the migrations to run, depending on your database size.

Added

- Allow user to change profile picture. (#151)

  Profile menu now has an extra option "Change picture". This allows uploading a new picture and optionally setting focus point for cropping a picture that is not square shape.
- Federate local profiles to remote followers on save. (#168)
- Process remote profiles entities on receive.

  Remote profiles were so far only created on first encounter. Now we also process incoming `Profile` entities from the federation layer.
- When following a remote profile, federate profile to them at the same time.
- It is now possible to expose statistics from a Socialhome node. This includes counts for users (total, 30 day, 6 month), local content and local replies. These will be exposed via the `NodeInfo` documents that for example the-federation.info node list consumes.

  By default statistics is off. Admins can switch the counts on by setting environment variable `SOCIALHOME_STATISTICS=True` and restarting Socialhome.
- Add user API token view. Allows retrieving an API token for usage in clients and tools. Allows also regenerating the token if it has been lost or exposed.

- Added bookmarklet to easily share external pages. The bookmarklet can be bookmarked from the 'Create' page. (#138)

  Sharing with the bookmarklet will copy the page url, title and optionally selected text into the create content text area. The bookmarklet is compatible with Diaspora, so for example the Firefox sharing service will work.

- Support receiving 'Share' entities. Show amount of shares on content. (#206)
- Show replies to shares on the original shared content. (#206)
- Add `share` endpoint to Content API. This enables creating and removing shares via the API. (#206)
- Allow sharing content. Clicking the share counter icon exposes a 'Share' button which when clicked will create a share. (#206)
- Allow unsharing content. Clicking the share counter icon exposes an 'Unshare' button (assuming the user has shared the content) which when clicked will remove the share. (#206)
- Federate local shares to remote nodes. (#206)
- There is now a 'My content' stream link in the navbar 'Streams' dropdown. This goes to your own profile all content stream.
- Add user preference for the new stream refactoring. If enabled, all streams that have a new version in progress will be rendered with the new frontend code based on Vue.js. (#202)

  Warning! The new frontent code doesn't have all the features of the current on yet.

- Content API has three new read only fields available:
  - `local`, boolean whether the content is local or remote.
  - `reply_count`, count of replies (including replies on shares)
  - `shares_count`, count of shares
- Make email notifications nicer by using HTML templates in addition to the plain text version. (#206)

  In addition to reply and follow notifications, send also when own content is shared.

Changed

- **Breaking change**. Content API results now return `visibility` as a string ('public', 'limited', 'site' or 'self'), not an integer.

Fixed

- There was no notification sent out when a local user followed a local user. This has now been fixed.

Removed

- **Breaking change**. Removed Content, Profile and Users API LIST routes. For now these are seen as not required for building a client and allow unnecessarily easy data mining.
- Removed content modal. Clicking timestamp in grid now directly loads the content detail view. (#162)

  Loading the content in a modal was an early experiment and didn't end out very usable.

- Removed reply button from replies. Technically, threaded replies are possible but the UI implementation is not done. Replying to a reply will be back once UI and federation layer will handle threaded replies properly.

7.13.6 0.3.1 (2017-08-06)

Fixed

- Bump `federation` library again to fix a regression in reply relaying due to security fixes in the library 0.14.0 release.

7.13.7 0.3.0 (2017-08-06)

Security

- Reject remote content updates via the federation layer which reference an already existing remote content object but have a different author.

  Note that locally created content was previously safe from this kind of takeover. This, even though serious, affects only remote created content stored locally.
- Reject remote reply updates via the federation layer which try to change the parent content reference.
- Bump federation to ensure remote entity authorship is verified correctly.

Added

- API has two new endpoints, the "Content" and "Image Upload" routes. (#120)
  - Content API allows browsing content objects that are visible to self, or public for anonymous users. Content objects owned by self can be updated or deleted. Creating content is also possible.
  - Image Upload API allows uploading images via the same mechanism that is used in the content create UI form. The uploaded image will be stored and a markdown string is passed back which can be added to content created in for example mobile clients. Note, uploading an image doesn't create any content itself, it just allows embedding images into content, just like in the UI.
- New API docs exposed by Django REST Swagger. These are in the same place as the old ones, at `/api/`. Adding to the documentation is still a work in progress.
- Add image upload button to the create/reply editor. This makes it possible to upload images from mobile browsers. (#120)
- Make profile "following" button link to "following contacts" page, if user is logged in and own profile.

Changed

- Create and update content will now redirect to the content created or updated. Previous behaviour was user preferred landing page.
- Delete content will now redirect back to the page where the delete was triggered from. Previous behaviour was user preferred landing page. If the content delete is triggered from the content detail page, redirect will happen to user preferred landing page as before. (#204)

Fixed

- Fix internal server error when replying to content that contained only characters outside the western Latin character sets.
- Visual fixes for content rendering in content delete page.

- Make direct profile handle search survive extra spaces before or after the searched handle.

7.13.8 0.2.1 (2017-07-30)

Fixed

- Fix reply form regression introduced in v0.2.0. (#217)

7.13.9 0.2.0 (2017-07-30)

Security

- Fix XSS vulnerability in profile edit. Unsanitized profile field input was allowed and one place showed a field without escaping it. The fields are now sanitized and escaping has been ensured.

  The problem concerned only local users and not remote profile fields which were correctly sanitized already.

Added

- Added search for profiles (#163)

  There is now a global search in the right side of the header. The search returns matches for local and remote profiles based on their name and username part of the handle. Profiles marked with visibility `Self` or `Limited` are excluded from the search results. Profiles marked with visibility `Site` will be excluded if not logged in, leaving only public profile results. If a direct match happens with a full handle, a redirect is done directly to the searched profile.

  IMPORTANT for node maintainers. After pulling in this change, you MUST run the command `python manage.py rebuild_index` to create the search index. Not doing this will cause an error to be raised when trying to search. The indexes are kept up to date automatically after running this command once.

- When searching for profiles based on handle, fetch profile from remote if it isn't found locally (#163)

Changed

- Improved content/reply create/edit form. Replies don't contain visibility or pinned form elements any more. Added also some help texts regarding drag'n'drop image embed, visibility and content pinning.

Fixed

- Make reply notifications to local users not send one single email with all local participants, but one email per participant. Previous implementation would have leaked emails of participants to other participants.
- Correctly send replies to remotes (#210)

  If parent content is local, send via the relayable forwarding mechanism. This ensures parent author signs the content. If parent author is remote, send just to the remote author. The remote author should then relay it.

- Ensure calling `Profile.private_key` or `Profile.key` don't crash if the profile doesn't have keys. Now the properties just return `None`.
- Fix regression in profile all content stream load more functionality. (#190)

- Filter out “limited” visibility profiles from API list results. These profiles are not available in the search so they shouldn’t be available to list through the API either.

7.13.10 0.1.0 (2017-07-27)

Initial versioned release. Main implemented features:

- Working streams (followed, public, profiles)
- Content creation
- Content OEmbed / OpenGraph previews
- Replies
- Follow/unfollow of profiles
- Contacts list
- Pinning content to profile