Socialhome Documentation

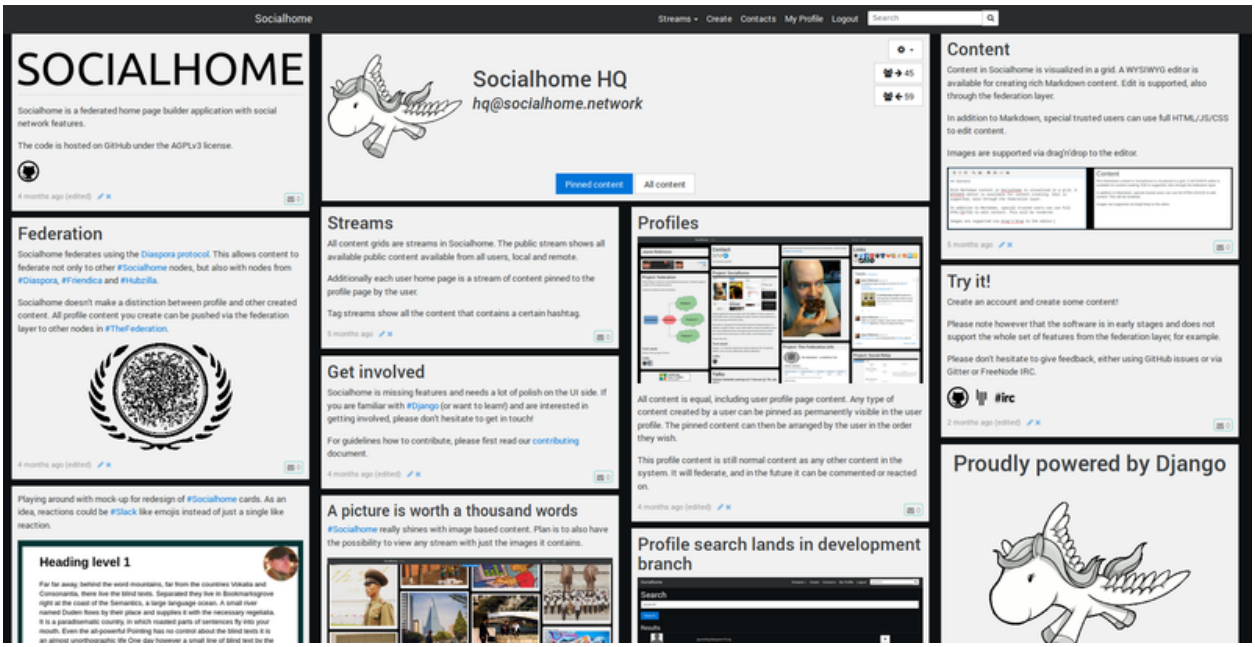
Release 0.5.0

Jason Robinson

Oct 07, 2017

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

Installation

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)

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.

Guides

See *Install guides*.

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)*

Ubuntu (14.04, manual)

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). Guide for non-upstart Ubuntu versions coming soon!

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.local` 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
SOCIALHOME_LOGFILE_FEDERATION=/home/socialhome/logs/socialhome-federation.log
```

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

Make the env file a bit less readable.

```
chmod 0600 env.local
```

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
node_modules/.bin/grunt build
python manage.py collectstatic
```

If you want to try the new Vue.js frontend, also do:

```
npm run dev
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
```

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*.

Ubuntu (14.04, Ansible)

See this Ansible role.

Other platforms

PR's welcome for guides for more platforms!

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!

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.

Change to Socialhome user

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

```
sudo su - socialhome
```

Activate virtualenv

```
workon socialhome
```

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
```

Install Python dependencies

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

```
pip-sync
```

Run migrations

```
python manage.py migrate
```

Install statics

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

If you want to try the new Vue.js frontend, also do:

```
npm run dev
python manage.py collectstatic
```

Restart the app

```
sudo service socialhome restart
```

Done!

Check the application and have fun!

Running an instance

Some notes on running a production instance.

Django admin

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

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
```

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.

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)
```

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.local` (assuming you are using this way to configure the application)
- The path `socialhome/media/` which contains for example image uploads

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.

Configuration

Configuration mainly happens through environment variables. Those are passed to Django via the file `env.local` 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_OPBEAT_ENABLE

Default: `False`

If you wish to enable Opbeat integration, set this to `True`. Also remember to set `DJANGO_OPBEAT_ORGANIZATION_ID`, `DJANGO_OPBEAT_APP_ID` and `DJANGO_OPBEAT_SECRET_TOKEN` to the values from Opbeat.

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_DOMAIN

Default: `socialhome.local`

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

SOCIALHOME_HTTPS

Default: `True`

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

SOCIALHOME_LOGFILE

Default: `/tmp/socialhome.log`

Where to write the main application log.

SOCIALHOME_LOGFILE_FEDERATION

Default: `/tmp/socialhome-federation.log`

Where to write the federation layer log.

SOCIALHOME_RELAY_DOMAIN

Default: `relay.iliketoast.net`

Which relay instance 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.

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.

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).

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.

Development

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

Clients

See the *Clients* section.

Clients

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

shcli

This is the official Python library and command line client.

See documentation and code on GitHub.

Community

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/).

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!

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

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
grunt dev
```

New Vue based frontend

The new Vue based frontend builds the bundles with Webpack. Execute the following to build the bundle.

```
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 `env.local` will be used automatically.

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

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.

Running tests

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
```

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
```

Building local documentation

```
cd docs
make html
```

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
```

Contact for help

See our communication channels in the *Community* page.

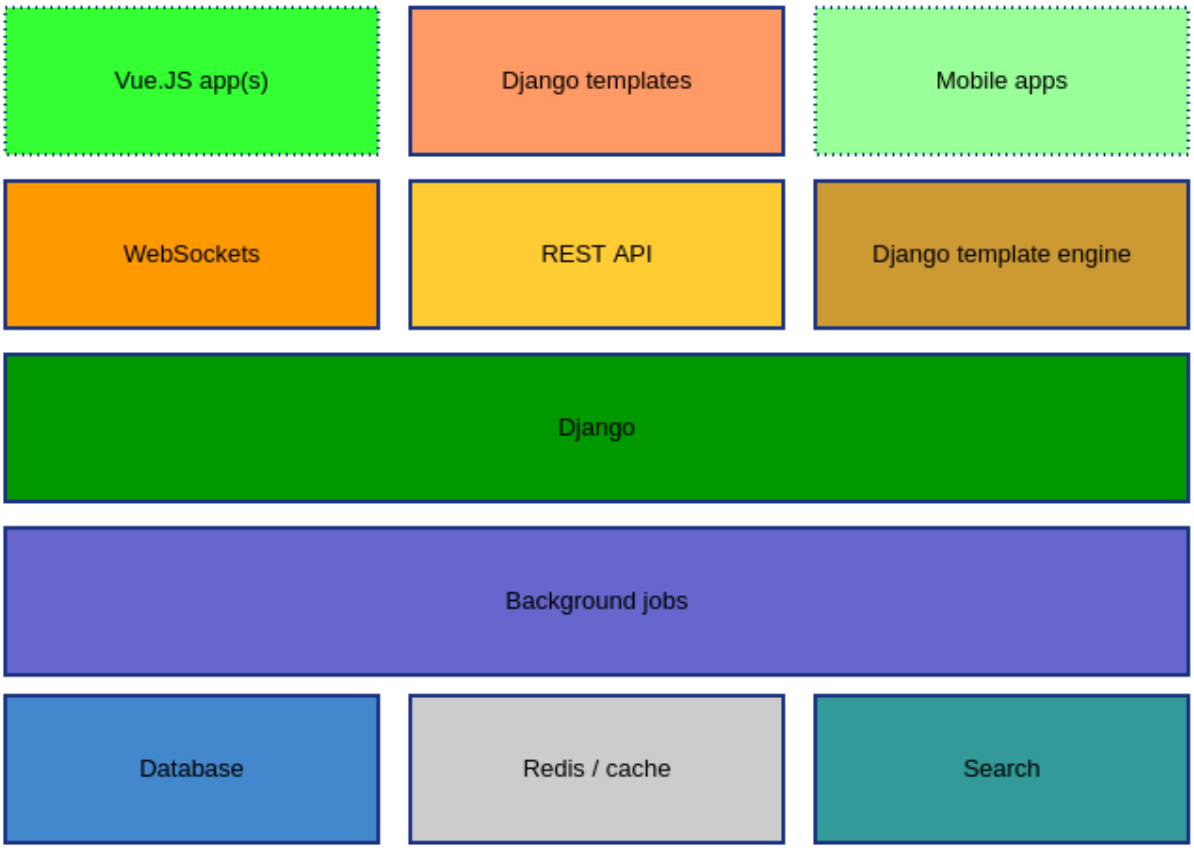
You can also ask questions or give feedback via issues.

Architecture

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

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.

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

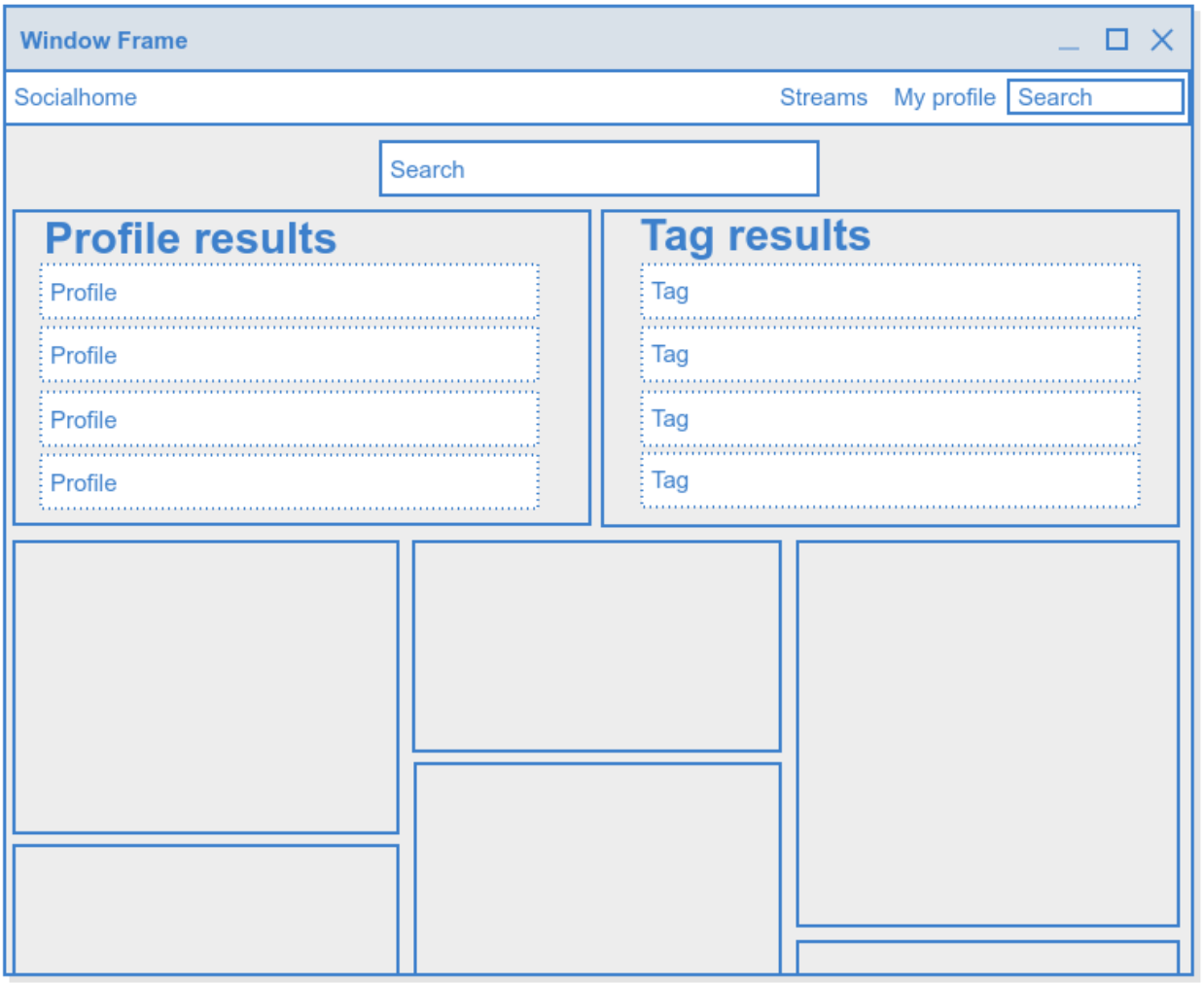
Indexes

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/static/js/streams.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.

Contributing

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

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.

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.

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.

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.

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

We use `py.test` as test runner but the tests themselves are Django based test classes. We have our own base class called `SocialhomeTestCase` which should be used as a base for all Django tests. Alternatively, to test Django class based views, you can use our `SocialhomeCBVTestCase`. Both these base classes inherit from the powerful django-test-plus test classes. Some old tests are pure py.test function based tests, feel free to convert these.

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.

Code style

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

There is an `.eslintrc` configuration provided as well. NOTE! This is only valid for the new Vue based frontend, not JS in `socialhome/static`.

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.

Changelog

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.

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.

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.

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.

0.2.1 (2017-07-30)

Fixed

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

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.

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