Hellolily

Release 0.4.30

April 25, 2016

Contents

CHAPTER 1

Overview

Hellolily is an open source CRM system built on Django and Angular.js.

CHAPTER 2

Community

Hellolily is being developed by Devhouse Spindle and a community of contributors. Check out the following resources for support and information:

- our IRC channel, #hellolily on irc.freenode.net

CHAPTER 3

Why Hellolily

- Unique
- Selling
- Points
- ...and much more

CHAPTER 4

Table of contents

4.1 Tutorials

The pages in this section of the documentation are aimed at the newcomer to Hellolily. They're designed to help you get started quickly, and show how easy it is to work with Hellolily as a developer who wants to customise it and get it working according to their own requirements.

These tutorials take you step-by-step through some key aspects of this work. They're not intended to explain the topics in depth, or provide reference material, but they will leave you with a good idea of what it's possible to achieve in just a few steps, and how to go about it.

Once you're familiar with the basics presented in these tutorials, you'll find the more in-depth coverage of the same topics in the Howto section.

The tutorials follow a logical progression, starting from installation of Hellolily and the creation of a brand new project, and build on each other, so it's recommended to work through them in the order presented here.

4.1.1 Installing Hellolily

Getting up and running a Hellolily instance is straightforward using our docker-compose setup.

Prerequisites

- you have docker and docker-compose installed
- you have git installed
- you have nodejs, npm and gulp installed

Installation

1. Checkout the Hellolily project and install gulp dependencies.

```
cd ~/projects/
git clone git@github.com:HelloLily/hellolily.git
cd hellolily
npm install
gulp build
```

2. Build the docker image. This takes a while the first time. [1]

```
docker-compose build
```

Note: This command needs to run every time the Dockerfile, requirements or patches are adjusted. Good practice would be to run it every time the git repo is updated. If nothing changed, the command would complete almost instantly.

3. Do a first time migration of the models.

```
docker-compose run web python manage.py migrate
```

4. Create a search index for ElasticSearch.

```
docker-compose run web python manage.py index
```

5. Populate the database with some testdata.

```
docker-compose run web python manage.py testdata
```

6. Run the Django development server along with dependent containers.

```
docker-compose run --service-ports web
```

Open http://localhost:8000 in your browser to see the running Hellolily instance. You can login using user **superuser1@lily.com** and **admin** as password. Congratulations, you now have Hellolily running!

If you prefer to configure Hellolily by hand, rather than using the docker installation, see Installing Hellolily by hand and then follow the rest of the tutorials.

Either way, you'll be able to find support and help from the numerous friendly members of the Hellolily community.

4.2 Howto guides

These guides presuppose some familiarity with Hellolily. They cover some of the same territory as the Tutorials, but in more detail.

4.2.1 Installing Hellolily by hand

Installing Hellolily with docker is recommended for most installations, but there may be cases where you find Docker not suitable. This installation describes how to install Hellolily without Docker.

Prerequisites

- you have python 2.7 and virtualenv installed
- you have git installed
- you have nodejs, npm and gulp installed
- you have postgresql up and running
- you have elasticsearch up and running
- you have rabbitmq up and running

[1] You might experience connectiviy issues duing build. Checkout this post for more info.

Installation

1. Make a virtualenv, checkout the Hellolily project and install gulp dependencies.

```
mkdir -p ~/projects/hellolily-env
cd ~/projects/hellolily-env
virtualenv2 .
. ./bin/activate
git clone git@github.com:HelloLily/hellolily.git
cd hellolily
npm install
gulp build
```

2. Install all related Python packages.

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

3. Setup a postgresql database for Hellolily.

```
sudo su postgres
cd ~
createdb hellolily
createuser hellolily -s
echo "ALTER USER hellolily WITH ENCRYPTED PASSWORD 'c2cg63&(e';" | psql
echo "create extension unaccent;" | psql
exit
```

4. Override settings using your own environment settings.

```
vim ~/projects/hellolily-env/hellolily/.env
```

```
DEBUG=1
MULTI_TENANT=1

DEFAULT_FROM_EMAIL=info@mydomain.org
SERVER_EMAIL=info@mydomain.org
EMAIL_USE_TLS=1
EMAIL_HOST=smtp.gmail.com
EMAIL_HOST_USER=info@mydomain.org
EMAIL_HOST_PASSWORD=
EMAIL_PORT=587

ADMINS=(('Yourname', 'your@email.com'),)

DATAPROVIDER_API_KEY=
# Make SECRET_KEY unique for your own site!
SECRET_KEY=abcdefghijklmnopqrstuvwxyz0123456789abcdefghijklmn

DATABASE_URL=postgres://hellolily:c2cg63&(e@localhost/hellolily

AWS_ACCESS_KEY_ID=aws_key
AWS_SECRET_ACCESS_KEY=aws_secret_access_key
AWS_STORAGE_BUCKET_NAME=aws_bucket_name

REDISTOGO_URL=redis://localhost:6379/0
CELERY_SEND_TASK_ERROR_EMAILS=0
```

5. Do a first time migration of the models.

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

6. Create a search index for ElasticSearch.

```
./manage.py index
```

7. Populate the database with some testdata.

```
./manage.py testdata
```

8. Run the Django development server.

```
./manage.py runserver 0:8000
```

Open http://localhost:8000 in your browser to see the running Hellolily instance. You can login using user **superuser1@lily.com** and **admin** as password. Congratulations, you now have the first part of Hellolily running!

Head over to the next topic to complete your Hellolily installation.

4.2.2 Setup GMail API

Hellolily uses Gmail and it's API to process email accounts. You need to register an account for the Google API and set the following properties in the *.env* file:

```
GA_CLIENT_ID=
GA_CLIENT_SECRET=
GMAIL_CALLBACK_URL=
```

On the Google API site you should enable the Gmail API and create an OAuth 2.0 client ID for a web application. That gives you the *GA_CLIENT_ID* and *GA_CLIENT_SECRET* tokens.

The *GMAIL_CALLBACK_URL* is http://domain:port/messaging/email/callback/ (e.g. http://127.0.0.1:8000/messaging/email/callback/).

4.2.3 Running and debugging Celery tasks

Celery is a distributed task queue in Python that's being used to perform several queued tasked from within the Hellolily platform. Think of tasks like syncing email and updating ElasticSearch indexes. Celery is already installed if you completed *the manual installation*.

Running

```
celery worker -B --app=lily.celery --loglevel=info -Q celery,queue1,queue2,queue3 -n beat.%h -c 1
```

Debugging

Code that's being executed by a Celery worker can be PDB'ed with RDB. Add the following to your Celery code:

```
from celery.contrib import rdb
rdb.set_trace()
```

You should see a notification in the Celery console when a worker stumbles upon the rbd trace. At that point you can use telnet to login to the remote PDB session like:

```
telnet localhost 6902
help
```

4.2.4 Running and writing tests

Good code needs tests. A project like Hellolily simply can't afford to incorporate new code that doesn't come with its own tests.

Tests provide some necessary minimum confidence: they can show the code will behave as it expected, and help identify what's going wrong if something breaks it. We certainly do want your contributions and fixes, but we need your tests with them too. Otherwise, we'd be compromising our codebase.

So, you are going to have to include tests if you want to contribute. However, writing tests is not particularly difficult, and there are plenty of examples in the code to help you.

Running tests

The testrunner accepts parameters to specify which tests you want to run:

Run specific tests by their stored id

```
./manage.py test --with-id
# run test #5 and 6 as assigned by --with-id
./manage.py test --with-id 5 6
```

Run tests by directory, filename, class or function

```
# all contacts tests.
./manage.py test ./lily/contacts/api/tests.py
# Run the test_create_object_with_relations test .
./manage.py test ./lily/contacts/api/tests.py:ContactTests:UnicodeTestCase.test_create_object_with_re
```

Run tests that are tagged by category

```
docker-compose run web ./manage test --attr=api
docker-compose run web ./manage test --attr=processes
```

Writing tests

The directory structure for test-related material for each Django app is:

```
./app/factories.py  <= contains the factory boy factories for this app's models
./app/tests/test_models.py  <= model functions are tested here (unit tests)
./app/tests/test_utils.py  <= helper functions from utils.py are tested here (unit tests)
./app/tests/test_views.py  <= contains ghostrunner tests.
./app/tests/tests.py  <= generic tests that don't fit anywhere else
```

Always inherit from a Hellolily testcases class. Modify the Hellolily testcase class if you need additional functionality. The first line of the test string should be short, concise and descriptive. Use the minimal amount of words to describe the test. Use the descriptive paragraph beneath the first docstring line to explain the test in detail.

What to test is an ungoing debate. In theory you should test as much as possible, but it's not practical to have tests for all changes, since it may take large amounts of time to test every detail. Here are some general guidelines in what situations testing may be useful:

1. User processes are ideal to write functional tests for. These tests are also a nice documentation source on how the platform is used.

Note:

- Override settings.DEBUG = True in your testcase to force DEBUG information.
- Check for settings.TESTING in your maincode if you need test-specific code.
- Exclude tests from running on production by checking for settings.PRODUCTION

Performance tips

- It's possible to reuse the testing database on the next run by executing tests with:

```
REUSE_DB=1 ./manage test
```

- Django nose supports running the testsuite with multiple processes. This is especially useful for functional tests.

```
./manage test --attr=views --processes=3 --process-timeout=600
```

4.3 Key topics

This section explains and analyses some key concepts in Hellolily. It's less concerned with explaining *how to do things* than with helping you understand *how it works*.

4.3.1 Email in Hellolily

Story about the Gmail API and why it rocks.

4.4 Reference

Technical reference material.

4.4.1 Configuration

4.4.2 Migrating in Lily

We use the built-in Django migrations for both our schema & data migrations. For more detailed information you should take a look at the Django docs.

Caveats

Data migrations have two things you need to know about:

- The modified date
- Seperating schema and data changes

Modified date

It is important to disable the updating of modified fields on models during data migrations. The modified date should always represent the date a **user** last modified it.

Unfortunately as of writing, there is no way to set this behaviour automatically, so you'll have to do it yourself using *update_modified = False*.

Practical example:

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.db import models, migrations


def data_migration_forward(apps, schema_editor):
    Account = apps.get_model('accounts', 'Account')

    for account in Account.objects.all():
        account.update_modified = False
        # Do some mutations on the account here
        account.save()  # Because update_modified is False, this will not update the modified date o

class Migration(migrations.Migration):
    dependencies = []

    operations = [
        migrations.RunPython(data_migration_forward),
    ]
```

Seperation of schema/data changes

You should **never** combine a schema and data migration in a single migration file. This is because of consistency and error prevention. In some cases it is possible to combine the two, but you still shouldn't do it.

```
Thus, on PostgreSQL, for example, you should avoid combining schema changes and RunPython operations
in the same migration or you may hit errors like OperationalError: cannot ALTER TABLE "mytable"
because it has pending trigger events.
```

As can be read on the Django docs somewhat near the end of the block.

<input value='asfdsdf' />

4.5 Development & community

Hellolily is an open-source project, and relies on its community of users to keep getting better.

4.5.1 Development of Hellolily

As an open source project, anyone is welcome to contribute in whatever form they are able, which can include taking part in discussions, filing bug reports, proposing improvements, contributing code or documentation, and testing the system - amongst others.

Devhouse Spindle

Hellolily is being developed at Devhouse Spindle by this team of core developers:

- Allard Stijnman https://github.com/snoepkast
- Cornelis Poppema https://github.com/cpoppema
- Ednan Pasagic https://github.com/epasagic
- Ferdy Galema https://github.com/ferdynice
- Jeroen van Veen https://github.com/jvanveen
- Luuk Hartsema https://github.com/luukhartsema
- Marco Vellinga https://github.com/m-vellinga
- Redmer Loen https://github.com/spindleredmer
- Stefan Strijker https://github.com/00stefan00
- Tom Offringa https://github.com/TomOffringa

4.5.2 Code of Conduct

Participation in the Hellolily project is governed by a code of conduct.

The Hellolily community is a pleasant one to be involved in for everyone, and we wish to keep it that way. Participants are expected to behave and communicate with others courteously and respectfully, whether online or in person, and to be welcoming, friendly and polite.

We will not tolerate abusive behaviour or language or any form of harassment.

Individuals whose behaviour is a cause for concern will be given a warning, and if necessary will be excluded from participation in official Hellolily channels (email lists, IRC channels, etc) and events.

Raising a concern

If you have a concern about the behaviour of any member of the Hellolily community, please contact one of the members of the core development team.

Your concerns will be taken seriously, treated as confidential and investigated. You will be informed, in writing and as promptly as possible, of the outcome.

4.6 Release notes & upgrade information

Some versions of Hellolily present more complex upgrade paths than others, and some **require** you to take action. It is strongly recommended to read the changelog notes carefully when upgrading. More extensive releases have their own release page linked from the changelog.

It goes without saying that you should **backup your database** before embarking on any process that makes changes to your database.

4.6.1 Changelog

0.4.30

- Gulp improvements

4.7 Indices and tables

- modindex
- genindex
- search