tarbena Documentation

Release tarbena

tarbena

Jun 17, 2019

Contents

CHAPTER 1

Contents:

1.1 Introduction

1.1.1 About the app

Each Town Hall in small towns needs to be introduced to new technologies. So for my people (Tárbena/Alicante) I have decided to make an application that suits their needs.

The sole purpose of `tarbena_app` is to add a *graphical user interface* system functionality and give you the ability to control | everything.

The key features of Tarbena apps are:

- Manage subsidies
- Manage parcels
- Manage keys that belongs to the Town Hall of Tarbena
- Add any other apps they think that can resolve their daily problems

1.1.2 Requirements

1. Django 1.11.6
2. Python 3.6.5
3. mysqlclient

1.1.3 Get ready (Windows)

- Download python
- Install `pip`. For windows we get it here
- Create an environment variable for pip:

```
setx PATH "%PATH%;C:\Python27\Scripts"
```

- Create an isolated environment for python with virtualenv:

```
pip install virtualenvwrapper-win
mkvirtualenv myproject
```

- Activate the virtualenv:

```
Scripts/activate => windows
source bin/activate => linux
```

- Install `Django` with pip
- Install `MySQL`:

```
For Python 2.7:
Download it here: http://www.codegood.com/download/10/
And with our virtualenv activated we do: easy_install file://C:/Users/ORDENADOR_1/
↪Downloads/MySQL-python-1.2.3.win32-py2.7.exe

For Python 3.6:
Download it here: https://www.lfd.uci.edu/~gohlke/pythonlibs/#mysql-python
32 bits refers to Python version and not to our system
```

- Create Django project and migrate the database:

```
django-admin startproject src
python manage.py migrate
python manage.py startapp subvenciones
```

1.1.4 Other ideas I want to add

1. Django compressor for minifying assets:

- django-compressor
- django-pipeline

2. New app 'terceros' with extended user model:

- Extend user model

3. Pip functionality with `pipenv`

- pipenv github functionality

4. Key app with QR code made with React

5. Library online app

1.2 Architecture

1.2.1 Configuration

1. Gitignore: local settings
2. Creating settings module with files for different environments: base, local and production
3. Creating project documentation with reStructuredText(.rst) files and Sphinx
4. Multiple requirements files

1.2.2 Apps

- **Authentication** (built-in django.auth): This app handles user signups, login, and logout
- **Profiles**: this app provides additional user profile information
- **Subvenciones**: subsidies management
- **Parcelas**: parcels management with Google Maps
- **Holidays**: holidays API with excluded days
- **Django Honeypot**: admin security
- **Django Admin Interface**: theme for Django Admin Panel
- **smart-selects**: area-ente selects functionality
- **django-notify-x**: django notification system activity
- **django-markdown-editor**
- **Favourites**: favourites of each user to appear in the index panel
- **Contracts**: this app manages contracts, contractors, generate pdfs
- **Gym**: town gym management
- **Museo**: town museu management
- **django-cleanup**: auto delete images when deleting object with realted image from the directories
- **django-autocomplete-light**: in the app parcelas auto search propietarios in the bbdd

Authentication

Created Login and Signup system in the root project (urls.py, templates, static).

Profiles

When a new user is created, his profile is also created.

Subvenciones

Application to manage subsidies.
Tables:

- Estado
- Colectivo
- Ente
- Area
- Subvencion
- Comment
- Like

Create Subvencion

When I create a new subsidie I add admin functionality in my form so I can add new fields suchs as: estado. Its a Django Popup functionality:
django-admin-popup-functionality

On the other hand, I insert the comment field into the subvencion form with a formset following this guide:
django formset implementation

Here for the formset field (contenido) I used markdown editor (added his configuration into settings):
Django markdown editor

In my custom template the editor didn't work as expected so in my base.html I had to add the following urls:

```
<link href="{% static 'plugins/css/ace.min.css' %}" type="text/css" media="all" rel=
↪"stylesheet" />
<link href="{% static 'plugins/css/semantic.min.css' %}" type="text/css" media="all"␣
↪rel="stylesheet" />
<link href="{% static 'plugins/css/resizable.min.css' %}" type="text/css" media="all"␣
↪rel="stylesheet" />
<link href="{% static 'martor/css/martor.min.css' %}" type="text/css" media="all" rel=
↪"stylesheet" />
<script type="text/javascript" src="{% static 'plugins/js/ace.js' %}"></script>
<script type="text/javascript" src="{% static 'plugins/js/semantic.min.js' %}"></
↪script>
<script type="text/javascript" src="{% static 'plugins/js/mode-markdown.js' %}"></
↪script>
<script type="text/javascript" src="{% static 'plugins/js/ext-language_tools.js' %}">
↪</script>
<script type="text/javascript" src="{% static 'plugins/js/theme-github.js' %}"></
↪script>
<script type="text/javascript" src="{% static 'plugins/js/highlight.min.js' %}"></
↪script>
<script type="text/javascript" src="{% static 'plugins/js/resizable.min.js' %}"></
↪script>
```

(continues on next page)

(continued from previous page)

```
<script type="text/javascript" src="{% static 'plugins/js/emojis.min.js' %}"></script>
<script type="text/javascript" src="{% static 'martor/js/martor.min.js' %}"></script>
```

Also I follow this guide to add the dropdown functionality for ente and area:
Dependent dropdown list

Datatables functionality to list subvenciones into the index.html

Notify user comment

For notifying the comments where user names appear I used this function:

```
def markdown_find_mentions(markdown_text, user, object):
    """
    To find the users that mentioned
    on markdown content using `BeautifulShoup`.

    input  : `markdown_text` or markdown content.
    return : `list` of usernames.
    """
    markdownify = import_string(MARTOR_MARKDOWNIFY_FUNCTION)
    mark = markdownify(markdown_text)
    soup = BeautifulSoup(mark, 'html.parser')
    markdown_users = list(set(
        username.text[1::] for username in
        soup.findAll('a', {'class': 'direct-mention-link'})
    ))

    all_users = User.objects.all()
    notify_list_users = []
    for a in all_users:
        if a.username in markdown_users:
            for o in User.objects.all().filter(username=a):
                notify_list_users.append(o)

    return notify.send(user, recipient_list=list(notify_list_users), actor=user,
                verb='comentarios', obj=object, target=object,
                nf_type='mention')
```

And then in the Created and UpdatedView in the formset:

```
comments_formset.save(commit=False)
    for f in comments_formset:
        contenido = f.cleaned_data.get("contenido")
        if contenido:
            # Notify comment
            markdown_find_mentions(self.request.POST['comments-0-contenido'], self.
↪request.user, self.object)
    comments_formset.save()
```

Filter subvenciones Index

For this purpose I used `django-filter`:

http://django-filter.readthedocs.io/en/latest/guide/install.html

Parcelas

Things you can do with this app:

- Find parcels by polygon in my municipality
- Add projects to your town hall (clean forests, roads . . .)
- Geolocation

Note: You can download KML related to that parcel

Holidays

When you create or edit subsidie there are two fields: start and end day (datefield). So if you put 13/07/2018 on the start day you will have some extra options such as +10, +20, +30 days. So to avoid holidays and weekends I created this app with Django REST framework (API).

Then in `create_subvenciones.js` I get the url with `$.getJSON` method.

Django Honeypot

https://github.com/jamesturk/django-honeypot

Django Admin Interface

https://djangopackages.org/grids/g/admin-styling/
https://github.com/fabiocaccamo/django-admin-interface
You can choose your own theme!

Django Cleanup

https://github.com/un1t/django-cleanup
Files or images are auto delted when object related to them is also deleted without any signals

Django Autocomplete Light v3

https://django-autocomplete-light.readthedocs.io/en/master/install.html
In the app parcelas I use it for ajax search propietarios when the user create new Parcela

Contracts

I use Adobe Acrobat PRO for creating powerfull PDFs with the custom fields (such as I did with parcelas app).

smart-select

https://github.com/digi604/django-smart-selects
I use this app for chaining selects (ente-area)

Installation:

```
pip install django-smart-selects
url(r'^chaining/', include('smart_selects.urls')), # into root url's, after admin
```

models.py:

```
from smart_selects.db_fields import ChainedForeignKey
area = ChainedForeignKey(
    Area,
    chained_field="ente",
    chained_model_field="ente",
    show_all=False,
    auto_choose=True,
    sort=True,
    default=''
)
```

Warning: In Lib/site-packages/smart_selects/static/smart_selects/admin/js/chainedfk.js has a problem, all his methods should be defined as object so I copy the new js from here: new chainedfk.js

And I copy it to my root static project so when I git pull to my production server I have it solved:
static/smart-selects/admin/js/chainedfk.js

And finally into my create.html and edit.html template I import them like this:

```
<script type="text/javascript" src="{% static 'smart-selects/admin/js/chainedfk.js'
↪%}"></script>
<script type="text/javascript" src="{% static 'smart-selects/admin/js/chainedm2m.js
↪' %}"></script>
<script type="text/javascript" src="{% static 'smart-selects/admin/js/bindfields.js
↪' %}"></script>
```

My old functionality is from here: old functionality

django-notify-x

https://github.com/v1k45/django-notify-x

```
pip install django-notify-x
INSTALLED_APPS = ('notify',)
url(r'^notifications/', include('notify.urls', 'notifications')),
python manage.py migrate notify
python manage.py collectstatic
```

Warning: `notify` application has in his models the verb to 50 limit character, just change it to TextField instead of CharField.

About the warning you can do:

```
# Lib/site-packages/notify/models.py
verb = models.TextField(verbose_name=_('Verb of the action'))
python manage.py makemigrations
python manage.py migrate
```

Views:

```
from notify.signals import notify
notify.send(self.request.user, recipient=self.request.user, actor=self.object,
                verb='subvención, %s' % (form.cleaned_data.get('nombre')), obj=self.
↪object,
                nf_type='create_subvencion')

Actor: The object which performed the activity.
Verb: The activity.
Object: The object on which activity was performed.
Target: The object where activity was performed.
```

django-markdown-editor (martor)

App used to create comments related to each subvencion. Besides, it allows you to add mentions to users with a custom query and then send them an email.

When you use in template:

```
comment.contenido|safe_markdown
```

This has in `Lib/site-packages/martor/extensions/mentions.py` this code:

```
def handleMatch(self, m):
username = self.unescape(m.group(2))

"""Makesure `username` is registered and actived."""
if MARTOR_ENABLE_CONFIGS['mention'] == 'true':
    if username in [u.username for u in User.objects.exclude(is_active=False)]:
        url = '{0}{1}/'.format(MARTOR_MARKDOWN_BASE_MENTION_URL, username)
        el = markdown.util.etree.Element('a')
        el.set('href', url)
        el.set('class', 'direct-mention-link')
        el.text = markdown.util.AtomicString('@' + username)
        return el
```

If you leave it like that you will have as many duplicated queries as mentions you have in that template. So to solve this, you just have to comment this line:

```
if username in [u.username for u in User.objects.exclude(is_active=False)]:
```

1.2.3 Project commands

To start the Python interactive interpreter with Django, using your `settings/local.py` settings file:

```
python manage.py shell --settings=tarbena.settings.local
```

To run the local development server with your `settings/local.py` settings file:

```
python manage.py runserver --settings=tarbena.settings.local
```

Backup my models:

```
python manage.py dumpdata myapp --indent=2 --output=myapp/fixtures/subsidies.json
python manage.py dumpdata auth --indent=2 --output=myapp/fixtures/auth.json
```

Load data from those backups:

```
python .\manage.py loaddata subsidies.json
```

Export my production database password and then get it or save it in a secure folder in the production server:

```
export MYSQL_PASSWORD=1234
'PASSWORD': os.getenv('MYSQL_PASSWORD'),
Or I can add it to my file and import it like the secret key and the email password.
```

Save my `SECREY_KEY` in a secure file in the production server:

```
>>> from django.core.signing import Signer
>>> signer = Signer()
>>> value = signer.sign('My string')
>>> value
'My string:GdMGD6HNQ_qdgxYP8yBZAdAIV1w'
```

1.2.4 Multiple requirements files

- **base.txt**: place the dependencies used in all environments
- **local.txt**: place the dependencies used in local environment such as debug toolbar
- **production.txt**: place the dependencies used in production environment
- **ci.txt** (continuous integration): the needs of a continuous integration such as django-jenkins or coverage

1.2.5 Admin Documentation

https://docs.djangoproject.com/en/1.11/ref/contrib/admin/admindocs/

```
pip install docutils
```

1.2.6 git-flow

The main branches

- **Master**
- **Develop**

I consider `origin/master` to be the main branch where the source code of HEAD always reflects a `production-ready state`.

I consider `origin/develop` to be the main branch where the source code of HEAD always reflects a state with the latest delivered development changes for the next release. Some would call this the `integration branch`.

Note:

When the source code in the develop branch reaches a stable point and is ready to be released, all of the changes should be merged back into `master` somehow and then tagged with a release number.

Therefore, each time when changes are merged back into master, this is a new production release by definition. We tend to be very strict at this, so that theoretically, we could use a Git hook script to automatically build and roll-out our software to our production servers everytime there was a commit on master.

Supporting branches

The different types of branches we may use are:

- **Feature branches**
- **Release branches**
- **Hotfix branches**

Feature branches

Comes from `develop` and must merge back into `develop`.
Branch naming convention: anything except `master`, `develop`, `release-*`, or `hotfix-*`

Feature branches (or sometimes called topic branches) are used to develop new features for the upcoming or a distant future release. When starting development of a feature, the target release in which this feature will be incorporated may well be unknown at that point. The essence of a feature branch is that it exists as long as the feature is in development, but will eventually be merged back into develop (to definitely add the new feature to the upcoming release) or discarded (in case of a disappointing experiment).

Feature branches typically exist in developer repos only, not in origin.

Creating a feature branch

```
$ git checkout -b myfeature develop
Switched to a new branch "myfeature"
```

Incorporating a finished feature on develop

```
$ git checkout develop
Switched to branch 'develop'

$ git merge --no-ff myfeature
Updating ea1b82a..05e9557
(Summary of changes)

$ git branch -d myfeature
Deleted branch myfeature (was 05e9557).

$ git push origin develop
```

Note: The `--no-ff` flag causes the merge to always create a new commit object, even if the merge could be performed with a fast-forward. This avoids losing information about the historical existence of a feature branch and groups together all commits that together added the feature.

Release branches

Comes from `develop` and must merge back into `develop` and `master`.
Branch naming convention: `release-*`

Release branches support preparation of a new production release.

Creating a release branch

Release branches are created from the develop branch. For example, say version 1.1.5 is the current production release and we have a big release coming up. The state of develop is ready for the "next release" and we have decided that this will become version 1.2 (rather than 1.1.6 or 2.0). So we branch off and give the release branch a name reflecting the new version number:

```
$ git checkout -b release-1.2 develop
Switched to a new branch "release-1.2"

$ ./bump-version.sh 1.2
Files modified successfully, version bumped to 1.2.

$ git commit -a -m "Bumped version number to 1.2"
[release-1.2 74d9424] Bumped version number to 1.2
1 files changed, 1 insertions(+), 1 deletions(-)
```

After creating a new branch and switching to it, we bump the version number. Here, bump-version.sh is a fictional shell script that changes some files in the working copy to reflect the new version. (This can of course be a manual change—the point being that some files change.) Then, the bumped version number is committed.

Finishing a release branch

When the state of the release branch is ready to become a real release, some actions need to be carried out. First, the release branch is merged into `master` (since every commit on `master` is a new release by definition, remember). Next, that commit on master must be tagged for easy future reference to this historical version. Finally, the changes made on the release branch need to be merged back into `develop`, so that future releases also contain these bug fixes.

```
$ git checkout master
Switched to branch 'master'

$ git merge --no-ff release-1.2
Merge made by recursive.
(Summary of changes)

$ git tag -a 1.2
```

Note: You might as well want to use the -s or -u <key> flags to sign your tag cryptographically.

To keep the changes made in the release branch, we need to merge those back into develop, though. In Git:

```
$ git checkout develop
Switched to branch 'develop'

$ git merge --no-ff release-1.2
Merge made by recursive.
(Summary of changes)
```

This step may well lead to a merge conflict (probably even, since we have changed the version number). If so, fix it and commit.

Now we are really done and the release branch may be removed, since we don't need it anymore:

```
$ git branch -d release-1.2
Deleted branch release-1.2 (was ff452fe).
```

Hotfix branches

Comes from `master` and must merge back into `develop` and `master`.
Branch naming convention: `hotfix-*`

Hotfix branches are very much like release branches in that they are also meant to prepare for a new production release, albeit unplanned. They arise from the necessity to act immediately upon an undesired state of a live production version. When a critical bug in a production version must be resolved immediately, a hotfix branch may be branched off from the corresponding tag on the master branch that marks the production version.

The essence is that work of team members (on the `develop` branch) can continue, while another person is preparing a quick production fix.

Creating the hotfix branch

Hotfix branches are created from the master branch. For example, say version 1.2 is the current production release running live and causing troubles due to a severe bug. But changes on develop are yet unstable. We may then branch off a hotfix branch and start fixing the problem:

```
$ git checkout -b hotfix-1.2.1 master
Switched to a new branch "hotfix-1.2.1"
$ ./bump-version.sh 1.2.1
Files modified successfully, version bumped to 1.2.1.
$ git commit -a -m "Bumped version number to 1.2.1"
[hotfix-1.2.1 41e61bb] Bumped version number to 1.2.1
1 files changed, 1 insertions(+), 1 deletions(-)
```

Don't forget to bump the version number after branching off! Then, fix the bug and commit the fix in one or more separate commits.

```
$ git commit -m "Fixed severe production problem"
[hotfix-1.2.1 abbe5d6] Fixed severe production problem
5 files changed, 32 insertions(+), 17 deletions(-)
```

Finishing a hotfix branch

When finished, the bugfix needs to be merged back into `master`, but also needs to be merged back into `develop`, in order to safeguard that the bugfix is included in the next release as well. This is completely similar to how release branches are finished.

First, update master and tag the release.

```
$ git checkout master
Switched to branch 'master'

$ git merge --no-ff hotfix-1.2.1
Merge made by recursive.
(Summary of changes)

$ git tag -a 1.2.1
```

Note: You might as well want to use the -s or -u <key> flags to sign your tag cryptographically.

Next, include the bugfix in develop, too:

```
$ git checkout develop
Switched to branch 'develop'

$ git merge --no-ff hotfix-1.2.1
```

(continues on next page)

(continued from previous page)

```
Merge made by recursive.
(Summary of changes)
```

The one exception to the rule here is that, *when a release branch currently exists, the hotfix changes need to be merged into that release branch, instead of** `develop`. Back-merging the bugfix into the release branch will eventually result in the bugfix being merged into develop too, when the release branch is finished. (If work in develop immediately requires this bugfix and cannot wait for the release branch to be finished, you may safely merge the bugfix into develop now already as well.)

Finally, remove the temporary branch:

```
$ git branch -d hotfix-1.2.1
Deleted branch hotfix-1.2.1 (was abbe5d6).
```

Note:

This work-flow guide I brought it from:
https://nvie.com/posts/a-successful-git-branching-model/
http://aprendegit.com/que-es-git-flow/

1.3 Deployment

1.3.1 Secret files

When deploying don't save the secret files into the project. Save them into a safe place and ignore them with gitignore. Things such as: database password, `SECRET_KEY`, email password, etc.

When I use:

```
install -r requirements/production.txt
```

Check that in base.txt I have no debug toolbar and it's in local.txt and also check that I have everything from development server with:

```
pip freeze -r requirements/base.txt
```

1.3.2 Notify app

Override verb character limit to TextField:

```
# Lib/site-packages/notify/models.py
verb = models.TextField(verbose_name=_('Verb of the action'))
python manage.py makemigrations
python manage.py migrate
```

1.3.3 django-markdown-editor (martor) app

App used to create comments related to each subvencion. Besides, it allows you to add mentions to users with a custom query and then send them an email.

When you use in template:

```
comment.contenido|safe_markdown
```

This has in `Lib/site-packages/martor/extensions/mentions.py` this code:

```
def handleMatch(self, m):
username = self.unescape(m.group(2))

"""Makesure `username` is registered and actived."""
if MARTOR_ENABLE_CONFIGS['mention'] == 'true':
    if username in [u.username for u in User.objects.exclude(is_active=False)]:
        url = '{0}{1}/'.format(MARTOR_MARKDOWN_BASE_MENTION_URL, username)
        el = markdown.util.etree.Element('a')
        el.set('href', url)
        el.set('class', 'direct-mention-link')
        el.text = markdown.util.AtomicString('@' + username)
        return el
```

If you leave it like that you will have as many duplicated queries as mentions you have in that template. So to solve this, you just have to comment this line:

```
if username in [u.username for u in User.objects.exclude(is_active=False)]:
```

1.3.4 PDF (WeasyPrint)

I used this package to generate pdfs from detailed subsidies.

For installing:

```
pip install WeasyPrint
```

> **Warning: Problems in Windows**:
>
> `OSError: dlopen() failed to load a library: cairo / cairo-2`
>
> To solve this I've got to install this: https://tschoonj.github.io/blog/2014/02/08/gtk2-64-bit-windows-runtime-environment-installer/

1.3.5 Pycairo error installation

When doing *pip install -r requirements/production.txt* it will give me an error: *pycairo failed building wheel*

To fix this in Linux:

```
sudo apt-get install libcairo2-dev libjpeg-dev libgif-dev
pip install pycairo
```

1.3.6 Mysql requirement production

Same as pycairo error. In Windows I install the mysql dependency from an exe but in Linux you should remove it from requirements when you install all of them and then install it by:

```
sudo apt-get install libmysqlclient-dev
```

1.3.7 Steps to deploy

1. In Ubuntu I can't install Mysql by my requirements file with the command. So I install this package first:

```
sudo apt-get install libmysqlclient-dev
# Then I can do:
pip install mysqlclient
```

2. Production settings I use:

```
'PASSWORD': os.getenv('MYSQL_PASSWORD'),

# Then I use this command:
export MYSQL_PASSWORD=asdasdasd23423

# So it gets the password from here
# Show all exported variables:
export -p

# Or search by one:
export -p | grep myvariable

# Delete an exported variable:
unset myvariable
```

Warning: This has an error and is that when you turn off your command line the variables gets deleted so you can save them into a secrete file into your server and get the info from there.

3. WeasyPrint error:

```
OSError: cannot load library 'pango-1.0': pango-1.0: cannot open shared object file:␣
↪No such file or directory.  Additionally, ctypes.util.find_library() did not manage␣
↪to locate a library called 'pango-1.0'

# With this error you just have to install:
apt-get install pango1.0-tests
```

4. The wsgi.py file: In python2 the execfile function works but in python3 it does not so you have to replace:

```
# This:
execfile(activate_this, dict(__file__=activate_this)) # py2

# For:
exec(open(activate_this).read()) #py3
```

5. In administration panel add SITES_ID: 1 (development) and 2 (production)
6. Add logging to production so you can debug in production server

7. Settings file. I create a dir named settings and inside of him I make him module by creating *__init__.py*. And inside that y import them. And I create each file for each environment but I ignore the local one so that in production it gets only the production file.

8. You can check your deployment and if you are using Apache you can check ur syntax:

```
python3 manage.py check --deploy
sudo apache2ctl configtest
```

9. Celery. In production server I need celery to run my tasks on the background. More info in my deploy file.

1.3.8 Celery

Now we have intstalled with pip in our project celery so its time to start it. Nice guides:

https://www.digitalocean.com/community/tutorials/how-to-use-celery-with-rabbitmq-to-queue-tasks-on-an-ubuntu-vps

https://thomassileo.name/blog/2012/08/20/how-to-keep-celery-running-with-supervisor/

We will include a "&" character at the end of our string to put our worker process in the background:

```
celery worker -A config &
celery -A config worker -l info

# This if I restart the server or if I close the server console by ssh stops working.
# So that I need a python program to keep runing celery in the background␣
↪(supervisor).
```

Keep in mind that I can't start celery as superadmin so I create a new user:

```
sudo adduser celery
whoami
# swap to new user
su - celery
# swap back to root
su -

# Giver superadmin permissions to new user
visudo
# I need to be superuser to modify this doc
# So after my root user I put celery and the same permissions as root

$ pip install supervisor # python 2
$ pip install git+https://github.com/Supervisor/supervisor # python 3
$ cd /path/to/your/project
$ echo_supervisord_conf > supervisord.conf

# Now swap back to celery user and go to my root project and activate my virtualenv
# Then run this command inside:
supervisord

# Next, just add this section after the [supervisord] section:
[program:celeryd]
command=/home/thomas/virtualenvs/yourvenv/bin/celery worker --app=myapp -l info
stdout_logfile=/path/to/your/logs/celeryd.log
```

(continues on next page)

(continued from previous page)

```
stderr_logfile=/path/to/your/logs/celeryd.log
autostart=true
autorestart=true
startsecs=10
stopwaitsecs=600
```

But this does not seem to work on the background with &, so I tried this new guide that works like a charm.

Guide I followed.

```
sudo apt-get install supervisor
```

In the root project create a folder called supervisor and inside create the two following files: | **subvenciones_celery.conf**

```
; ==================================
;  celery worker supervisor example
; ==================================

; the name of your supervisord program
[program:configcelery]

; Set full path to celery program if using virtualenv
command=/home/admin/tarbena/bin/celery worker -A config --loglevel=INFO

; The directory to your Django project
directory=/home/admin/tarbena

; If supervisord is run as the root user, switch users to this UNIX user account
; before doing any processing.
user=amos

; Supervisor will start as many instances of this program as named by numprocs
numprocs=1

; Put process stdout output in this file
stdout_logfile=/var/log/celery/subvenciones_worker.log

; Put process stderr output in this file
stderr_logfile=/var/log/celery/subvenciones_worker.log

; If true, this program will start automatically when supervisord is started
autostart=true

; May be one of false, unexpected, or true. If false, the process will never
; be autorestarted. If unexpected, the process will be restart when the program
; exits with an exit code that is not one of the exit codes associated with this
; process' configuration (see exitcodes). If true, the process will be
; unconditionally restarted when it exits, without regard to its exit code.
autorestart=true

; The total number of seconds which the program needs to stay running after
; a startup to consider the start successful.
startsecs=10
```

(continues on next page)

(continued from previous page)

```
; Need to wait for currently executing tasks to finish at shutdown.
; Increase this if you have very long running tasks.
```

subvenciones_celerybeat.conf

```
==============================
;  celery beat supervisor example
; ==============================

; the name of your supervisord program
[program:configcelerybeat]

; Set full path to celery program if using virtualenv
command=/home/admin/tarbena/bin/celerybeat -A config --loglevel=INFO

; The directory to your Django project
directory=/home/admin/tarbena/src

; If supervisord is run as the root user, switch users to this UNIX user account
; before doing any processing.
user=amos

; Supervisor will start as many instances of this program as named by numprocs
numprocs=1

; Put process stdout output in this file
stdout_logfile=/var/log/celery/subvenciones_beat.log

; Put process stderr output in this file
stderr_logfile=/var/log/celery/subvenciones_beat.log

; If true, this program will start automatically when supervisord is started
autostart=true

; May be one of false, unexpected, or true. If false, the process will never
; be autorestarted. If unexpected, the process will be restart when the program
; exits with an exit code that is not one of the exit codes associated with this
; process' configuration (see exitcodes). If true, the process will be
; unconditionally restarted when it exits, without regard to its exit code.
autorestart=true

; The total number of seconds which the program needs to stay running after
; a startup to consider the start successful.
startsecs=10

; if your broker is supervised, set its priority higher
; so it starts first
priority=999
```

Then create the logs in */var/log*. Now just copy these files to the remote server in the */etc/supervisor/conf.d/* directory. Then:

```
$ sudo supervisorctl reread
$ sudo supervisorctl update

$ sudo supervisorctl stop configcelery
$ sudo supervisorctl start configcelery
$ sudo supervisorctl status configcelery
```

The default Celery scheduler creates some files to store its schedule locally. These files would be *celerybeat-schedule.db* and *celerybeat.pid*. If you are using a version control system like Git (which you should!), it is a good idea to ignore this files and not add them to your repository since they are for running processes locally.

1.3.9 Varnish Cache

HTTP accelerator designed for content-heavy dynamic web sites as well as APIs. So that make our web faster.

1.4 Celery

Everything you execute in a view will affect response time. In many situations you might want to return a response to the user as quickly as possible and let the server execute some process asynchronously.

One example is sending e-mails to users. If your site sends e-mail notifications from a view, the SMTP connection might fail or slow down the response. Launching asynchronous tasks is essential to avoid blocking execution.

1.4.1 Installation

```
pip install celery==3.1.18
```

Note: Celery requires a *message broker* in order to handle requests from an external source. The broker takes care of sending messages to Celery *workers*, which process tasks as they receive them. Let's install a message broker.

1.4.2 Message Broker: RabbitMQ or Redis

There are several options to choose as a message broker for Celery, including key-value stores such as Redis or an actual message system such as RabbitMQ. We will configure Celery with RabbitMQ, since it's a recommended message worker for Celery.

If you are using Linux, you can install RabbitMQ from the shell using the following command

```
apt-get install rabbitmq
```

If you need to install RabbitMQ on Mac OS X or Windows, you can find standalone versions at https://www.rabbitmq.com/download.html .

First you need to download and install erlang: http://www.erlang.org/downloads

After installing it, launch RabbitMQ using the following command from the shell:

```
rabbitmq-server
```

You will see an output that ends with the following line:

```
Starting broker... completed with 10 plugins.
```

RabbitMQ is running and ready to receive messages.

1.4.3 Adding Celery to my project

You have to provide a configuration for the Celery instance. Create a new file next to the *settings.py* name it *celery.py*. This file will contain the Celery configuration for your project.

Now you can add asynchronous tasks to your project.

1.4.4 Monitoring Celery

You might want to monitor the asynchronous tasks that are executed. Flower is a web-based tool for monitoring Celery. You can install Flower using the command

```
pip install flower
```

Once installed, you can launch Flower running the following command from your project directory:

```
celery -A config flower
```

Open http://localhost:5555/dashboard in your browser. You will be able to see the active Celery workers and asynchronous tasks' statistics. The above command will tell you the tasks you have.

Another command for check if the tasks are successfully launched is.

```
celery -A config worker -l info
```

Things I have created:

- celery.py # in my config folder
- tasks.py # in the app you want to run tasks
- then in views you define that task

1.5 Crontab

The cron daemon is a long-running process that executes commands at specific dates and times. You can use this to schedule activities, either as one-time events or as recurring tasks.

1.5.1 Usage

I use it for making a database backup. First I need to activate my virtualenvironment so I create a bash script like this:

```
# backup.sh
#!/bin/bash
# Script - Backup my app
source /home/admin/tarbena/bin/activate
python /home/backup2.py
```

What I do here is activate my virtualenv and then I execute my python script that contains the command for generating the json with my data and send an email with the backup:

```
import os
import subprocess
import smtplib
import datetime
import os.path as op
from email.mime.multipart import MIMEMultipart
from email.mime.base import MIMEBase
from email.mime.text import MIMEText
from email.utils import COMMASPACE, formatdate
from email import encoders

def send_mail(send_from, send_to, subject, message, files=[],
              server="smtp.gmail.com", port=587, username='myemail@gmail.com',␣
↪password='myemailpassword',
              use_tls=True):
    """Compose and send email with provided info and attachments.

    Args:
        send_from (str): from name
        send_to (str): to name
        subject (str): message title
        message (str): message body
        files (list[str]): list of file paths to be attached to email
        server (str): mail server host name
        port (int): port number
        username (str): server auth username
        password (str): server auth password
        use_tls (bool): use TLS mode
    """
    msg = MIMEMultipart()
    msg['From'] = send_from
    msg['To'] = send_to
    msg['Date'] = formatdate(localtime=True)
    msg['Subject'] = subject

    msg.attach(MIMEText(message))

    for path in files:
        part = MIMEBase('application', "octet-stream")
        with open(path, 'rb') as file:
            part.set_payload(file.read())
        encoders.encode_base64(part)
        part.add_header('Content-Disposition',
                        'attachment; filename="{}"'.format(op.basename(path)))
        msg.attach(part)
```

(continues on next page)

(continued from previous page)

```
    smtp = smtplib.SMTP(server, port)
    if use_tls:
        smtp.starttls()
    smtp.login(username, password)
    smtp.sendmail(send_from, send_to, msg.as_string())
    smtp.quit()

my_file = "/home/subsidies.json"

if os.path.exists(my_file):
        os.remove("/home/subsidies.json")

subprocess.call(["python", "/home/admin/tarbena/src/manage.py", "dumpdata",
↪"subvenciones", "--indent=2", "--output=/home/subsidies.json"])

now = datetime.datetime.now()
send_mail('agoraweb700@gmail.com', ", ".join(['amosisa700@gmail.com',
↪'jctarbena@gmail.com']), 'Tarbena app DB Backup', 'Tarbena app DB Backup a fecha de
↪{0}'.format(now), ['/home/subsidies.json'])
```

Note: To make email work I need to install email and don't name ur script like *email.py*. *pip install email*. Also give ur scripts permissions to be executed: *chmod ugo+x backup.sh (the same with backup2.py or do it by reference chmod –reference=backup.sh backup2.py)*.

1.5.2 Crontab commands

```
crontab -e # it opens the editor and at the bottom you can add your task to repeat

# you can add it like so
*/5 * * * * /home/backup.sh # it will repeat the script every 5mins

crontab -l # to list your repeating scripts added
```

Good page to calculate your repeating time: every monday, every 5mins, etc

1.6 Biblioteca

Installation of Koha, Integrated Library System.

1.6.1 Installation

Memcached error

https://openschoolsolutions.org/how-to-install-and-set-up-koha-for-schools-part-1/

https://wiki.koha-community.org/wiki/Koha_on_ubuntu_-_packages#Introduction

https://wiki.koha-community.org/wiki/Koha_on_ubuntu_-_packages#Appendix_A:_Named-based_vs._IP-based_installations

https://wiki.koha-community.org/wiki/Commands_provided_by_the_Debian_packages#koha-disable

https://wiki.koha-community.org/wiki/Koha_en_Ubuntu_-_paquetes#Configurando_Apache

CHAPTER 2

Indices and tables

- genindex
- modindex
- search