Django Cast Documentation

Release 0.1.30

Jochen Wersdörfer

Feb 27, 2022

Contents

Contents:

CHAPTER 1

Django Cast

Just another blogging / podcasting package

1.1 Documentation

The full documentation is at https://django-cast.readthedocs.io.

1.2 Installation Screencast

1.3 Quickstart

Install Django Cast:

```
pip install django-cast
```

Add django-cast and some dependencies to your INSTALLED_APPS:

```
INSTALLED_APPS = (
    ...
    "django.contrib.sites",
    "imagekit",
    "ckeditor",
    "ckeditor_uploader",
    "crispy_forms",
    "django_filters",
```

(continues on next page)

(continued from previous page)

```
    "rest_framework",
    "rest_framework.authtoken",
    "filepond.apps.FilepondConfig",
    "cast.apps.CastConfig",
    "watson",
    "fluent_comments",
    "threadedcomments",
    "django_comments",
    ...
)

SITE_ID = 1
```

Add required settings:

```
# CKEditor
CKEDITOR_UPLOAD_PATH = "uploads/ckeditor/"
CKEDITOR_IMAGE_BACKEND = "pillow"
AWS_QUERYSTRING_AUTH = False
X_FRAME_OPTIONS = "SAMEORIGIN"
CKEDITOR_CONFIGS = {
        "default": {
        "removePlugins": "stylesheetparser",
        "allowedContent": True,
        "enterMode": 2,
    },
}

# REST
REST_FRAMEWORK = {
    # Use Django's standard django.contrib.auth permissions,
    # or allow read-only access for unauthenticated users.
    "DEFAULT_AUTHENTICATION_CLASSES": (
        "rest_framework.authentication.SessionAuthentication",
        "rest_framework.authentication.TokenAuthentication",
    )
}

# django imagekit
IMAGEKIT_DEFAULT_CACHEFILE_STRATEGY="imagekit.cachefiles.strategies.Optimistic"

# Comments
COMMENTS_APP = 'fluent_comments'
FLUENT_COMMENTS_EXCLUDE_FIELDS = ('email', 'url', "title")
CAST_COMMENTS_ENABLED = True
```

Add Django Cast's URL patterns:

```
from django.urls import include, path, re_path

from rest_framework.documentation import include_docs_urls
from rest_framework.authtoken import views as authtokenviews


urlpatterns = [
    ...
    # Cast urls
```

(continues on next page)

(continued from previous page)

```
    path("api/api-token-auth/", authtokenviews.obtain_auth_token),
    path("docs/", include_docs_urls(title="API service")),
    path("ckeditor/", include("ckeditor_uploader.urls")),
    # Uploads
    path("uploads/", include("filepond.urls", namespace="filepond")),
    # Cast
    path("cast/", include("cast.urls", namespace="cast")),
    # Threadedcomments
    re_path(r'^cast/comments/', include('fluent_comments.urls')),
    ...
]
```

The api token auth urls and the docs urls are both necessary to provide api endpoints with the right namespace. The django-filepond app is used to dispatch uploads to the right media models.

1.4 Features Overview

- Support for responsive images / video / audio media objects
- Use django template syntax for posts allowing you to use custom template tags for galleries etc. for example
- Good looking file uploads via filepond
- Chaptermarks for podcast Episodes
- Fulltext search via django-watson
- Faceted navigation via django-filter
- Comments for posts via django-contrib-comments, django-threadedcomments and django-fluent-comments

1.5 Running Tests

1.5.1 Install Dependencies

Non python packages that are required but need to be installed using your operating system package manager:

- ffmpeg

Install packages that are required to be able to run the tests via poetry:

```
$ poetry install
```

1.5.2 Run Tests

Does the code actually work?

```
$ poetry shell
$ python runtests.py tests
```

1.6 Credits

Tools used in rendering this package:

- django-imagekit
- filepond
- django-filter
- django-watson
- django-contrib-comments
- django-threadedcomments
- django-fluent-comments
- podlove-web-player
- podlove-subscribe-button
- djangorestframework
- django-model-utils
- django-crispy-forms
- django-ckeditor
- Cookiecutter
- cookiecutter-djangopackage
- jquery
- bootstrap

CHAPTER 2

Features

2.1 Comments

You can enable / disable comments on app, blog and post-level. For app-level, there's a global switch you can use in the settings. Blog and post models have a comments_enabled database field. They are set to True by default.

2.1.1 Settings

```
# Switch to enable/disable comments globally. By default it's False
CAST_COMMENTS_ENABLED = True
```

2.1.2 Caveats

The ajax-calls django-fluent-comments does depend on the availability of a full jquery version. The min-version shipped by cookiecutter-django is not sufficient, therefore an additional jquery version is loaded on the post detail page when comments are enabled.

2.2 Content

The content of a blog post is just a normal django template. There are some special templatetags and context variables, though.

2.2.1 Templatetags

There are some templatetags to include media models from uploads into your post content. If you use the javascript editing frontend the primary keys for media models will be set automatically.

Image

To include an image, simply use the image templatetag:

```
{% image 1 %}
```

The number denotes the primary key of the image model. The templatetag will handle the srset attribute for you automatically.

Gallery

If you have more than one image you want to display, there's the gallery templatetag:

```
{% gallery 1 %}
```

The number denotes the primary key of the gallery model you want to include. The tag will build a modal bootstrap dialog to show the included images as well as setting the srcset attributes for the modally displayed images and the thumbnails.

Video

To include an video, simply use the video templatetag:

```
{% video 1 %}
```

The number denotes the primary key of the video model. The templatetag will handle the poster attribute for you automatically.

Audio

To include an audio model, simply use the audio templatetag:

```
{% audio 1 %}
```

The number denotes the primary key of the audio model. The templatetag won't do that much, because the player is pure javascript and chaptermarks and other metadata are pulled from a rest-api by the player. Note that you have to explicitly set the podcast_audio if you want some audio model included as the podcast episode audio. There can only be one such model whereas you can link to an arbitrary number of audio models that are not the podcast episode audio.

2.2.2 Content in post list and detail view

If you want some content only to be visible on the post detail page, just wrap it with an if-tag that evaluates a contenxt varible set by the list/detail view:

```
{% if include_detail %}
    This content will only be visible on the post detail page.
{% endif  %}
```

This might be useful for long shownotes-sections you have sometimes for podcast episodes etc..

2.3 Templates

There are some ready to use templates included. For example to display the list of posts or the detail page of a post. Often there are blocks that can be overwritten to customize the appearance of those templates.

2.3.1 Example Podcast

Feeds

For a podcast like python-podcast.de you may want to add the podlove-subscribe-button instead of having the default feed icon. This is quite easy to accomplish. Just create a template named templates/cast/post_list.html and set the content to:

```
{% extends "cast/post_list.html" %}
{% block feeds %}
<p>
<script>window.podcastData={"title":"Python Podcast","subtitle":"Ein␣
↪deutschsprachiger Podcast rund um die Programmiersprache Python","description":"",
↪"cover":"https://d2mmy4gxasde9x.cloudfront.net/cast_images/itunes_artwork/pp_itunes_
↪artwork_3k.png","feeds":[{"type":"audio","format":"aac","url":"https://python-
↪podcast.de/show/feed/podcast/m4a/rss.xml","directory-url-itunes":"https://podcasts.
↪apple.com/de/podcast/python-podcast/id1445331513"},{"type":"audio","format":"mp3",
↪"url":"https://python-podcast.de/show/feed/podcast/mp3/rss.xml","directory-url-
↪itunes":"https://podcasts.apple.com/de/podcast/python-podcast/id1445331513"},{"type
↪":"audio","format":"ogg","url":"https://python-podcast.de/show/feed/podcast/oga/rss.
↪xml","directory-url-itunes":"https://podcasts.apple.com/de/podcast/python-podcast/
↪id1445331513"},{"type":"audio","format":"opus","url":"https://python-podcast.de/
↪show/feed/podcast/opus/rss.xml","directory-url-itunes":"https://podcasts.apple.com/
↪de/podcast/python-podcast/id1445331513"}]}</script><script class="podlove-subscribe-
↪button" src="https://cdn.podlove.org/subscribe-button/javascripts/app.js" data-
↪language="de" data-size="big" data-json-data="podcastData" data-color="#469cd1"␣
↪data-format="cover" data-style="filled"></script><noscript><a href="https://python-
↪podcast.de/show/feed/podcast/m4a/rss.xml">Subscribe to feed</a></noscript>
</p>
{% endblock feeds %}
```

The javascript snipped was generated by the subscribe-button-generator which was also really easy.

Post Detail Link

Or maybe you want to overwrite the default post detail link if you used "{% if include_detail %}" to exclude the shownotes of your podcast from the episode list view.

```
{% extends "cast/post_list.html" %}
{% load i18n %}

{% block detail_link %}
<a href="{% url "cast:post_detail" blog_slug=blog.slug slug=post.slug %}">
  Shownotes | {% trans "Comments" %} | Permalink
</a>
{% endblock detail_link %}
```

CHAPTER 3

Deployment

Here we describe how to deploy a project named "foobar" to production. We are using cookiecutter-django for convenience, but this should also be possible with other project bootstrapping mechanisms.

Contents:

3.1 Locally / Setting up your development machine

Install virtualenvwrapper and create a virtual environment for your project:

```
mkvirtualenv -p /usr/local/bin/python3 foobar
```

Install cookiecutter into your newly create virtual environment:

```
pip install cookiecutter
```

Use the cookiecutter-django template to bootstrap your project:

```
cookiecutter https://github.com/pydanny/cookiecutter-django
```

Don't forget to activate the options for Docker or Heroku if you plan to use them. And set the "use whitenoise" configutation option to "yes" because this will get your static file serving on heroku work without any additional config. Saying "no" to this, will try to use aws S3, which I couldn't get to work (see below).

Enter your new project directory and checkin your first commit:

```
cd foobar
git init
git add .
git commit -m "first commit"
```

Optionally you can associate your project dir with a github repository (change url to match your username/reponame):

```
git remote add origin git@github.com:your_username/foobar.git
git push -u origin master
```

3.1.1 Running the App locally

You could also use docker for this, but for now let's run the development server locally. At first, add the "django-cast" requirement to your base.txt requirements file and then install all the required packages into your virtualenv:

```
echo "django-cast" >> requirements/base.txt
pip install -r requirements/local.txt
```

You should already have a locally installed postgres server up and running. Ok, now let's create the required database user, the database and all its tables. It's also very useful to create a django superuser right away:

```
createdb foobar;createuser foobar; psql -d foobar -c "GRANT ALL PRIVILEGES ON␣
↪DATABASE foobar to foobar;"
./manage.py migrate
./manage.py createsuperuser
```

Now you should be able to start your development server locally and see an empty page:

```
./manage.py runserver_plus 0:8000
open http://localhost:8000/
```

Your development server should now be reachable at http://localhost:8000

Open only works on mac OS, but you can just point your browser to this url. You should be able to sign in with your superuser account in the django admin. If you want to sign in regularily, you have to paste the confirmation url shown on the dev-server console when you try to sign in.

3.1.2 Installation using Docker

Install:

- Docker for your OS
- docker-compose

You need to have set the docker option to "yes" when you created the project diretory.

```
docker-compose -f local.yml build
docker-compose -f local.yml run django ./manage.py migrate
docker-compose -f local.yml up
```

Your development server should now also be reachable at http://localhost:8000

3.2 Useful third party services for production

There are some services that might be useful or even required when you run a website. Being able to send mail for example is quite useful if you want to send newly registered users a confirmation link.

3.2.1 Mailgun

If you use *mailgun* as an email service you have to register a mailgun account and set up your dns records accordingly. One caveat: If you use the eu region you have to change your base api url in "config/settings/production.py" to:

```
"MAILGUN_API_URL": env("MAILGUN_API_URL", default="https://api.eu.mailgun.net/v3"),
```

3.2.2 Sentry

This is the place where tracebacks that occured on the production system get recorded. You'll need to signup for an account.

3.2.3 Amazon S3

You'll probably use S3 for storing uploaded files and for your MEDIA_ROOT. .. mailgun: https://mailgun.com

3.3 Heroku

3.3.1 Install the heroku command line app

At first you have to create an heroku account and install the heroku command line app.

Then create your app with the heroku client and make your newly created app the default app, to avoid having to specify it for every heroku toolbelt call with "-a":

```
heroku create --buildpack https://github.com/heroku/heroku-buildpack-python --region␣
↪eu
heroku git:remote -a <name-of-the-app>
```

3.3.2 Use S3 for storing media files

You probably want to use S3 to store your media files (user uploaded content, images for blog posts etc). We use django-imagekit for responsive images and there is some incompatibility between boto and django-imagekit keeping it from working out of the box. Luckily there's a workaround. At this custom storage class to your "config/settings/production.py" file and use it:

```
import os
from tempfile import SpooledTemporaryFile
...

class CustomS3Boto3Storage(S3Boto3Storage):
        """
        This is our custom version of S3Boto3Storage that fixes a bug in
        boto3 where the passed in file is closed upon upload.

        https://github.com/boto/boto3/issues/929
        https://github.com/matthewwithanm/django-imagekit/issues/391
        """

        location = "media"
```

(continues on next page)

(continued from previous page)

```
        file_overwrite = False
        default_acl = "public-read"

        def _save_content(self, obj, content, parameters):
                """
                We create a clone of the content file as when this is passed to boto3
                it wrongly closes the file upon upload where as the storage backend
                expects it to still be open
                """
                # Seek our content back to the start
                content.seek(0, os.SEEK_SET)

                # Create a temporary file that will write to disk after a specified␣
↪size
                content_autoclose = SpooledTemporaryFile()

                # Write our original content into our copy that will be closed by␣
↪boto3
                content_autoclose.write(content.read())

                # Upload the object which will auto close the content_autoclose␣
↪instance
                super(CustomS3Boto3Storage, self)._save_content(
                        obj, content_autoclose, parameters
                )

                # Cleanup if this is fixed upstream our duplicate should always close
                if not content_autoclose.closed:
                        content_autoclose.close()

...
DEFAULT_FILE_STORAGE = "config.settings.production.CustomS3Boto3Storage"
```

Using S3 in a non-default region

If you want to use S3 in the region “eu-central-1” you have to set some additional parameters in your “config/settings/production.py”:

```
AWS_AUTO_CREATE_BUCKET = True
AWS_S3_REGION_NAME = 'eu-central-1'  # if your region differs from default
AWS_S3_SIGNATURE_VERSION = 's3v4'
AWS_S3_FILE_OVERWRITE = True
```

Using cloudfront as CDN

If you want to deliver your media files via cloudfront there’s an additional option you’ll have to set:

```
AWS_S3_CUSTOM_DOMAIN = env('CLOUDFRONT_DOMAIN')
```

3.3.3 Set the configuration variables for heroku

Now we have to setup some heroku specific stuff. For some of the addons you might have to add credit card information to your heroku account:

```
heroku addons:create heroku-postgresql:hobby-dev
heroku pg:backups schedule --at '02:00 Europe/Berlin' DATABASE_URL
heroku addons:create heroku-redis:hobby-dev
heroku addons:create mailgun:starter
heroku config:set PYTHONHASHSEED=random
heroku config:set WEB_CONCURRENCY=4
heroku config:set DJANGO_DEBUG=False
heroku config:set DJANGO_SETTINGS_MODULE=config.settings.production
heroku config:set DJANGO_SECRET_KEY="$(openssl rand -base64 64)"
heroku config:set DJANGO_ADMIN_URL="$(openssl rand -base64 4096 | tr -dc 'A-HJ-NP-Za-
↪km-z2-9' | head -c 32)/"
# use your own app name here..
heroku config:set DJANGO_ALLOWED_HOSTS=<your_app_name>.herokuapp.com
heroku config:set DJANGO_AWS_ACCESS_KEY_ID=<your_aws_key_id>
heroku config:set DJANGO_AWS_SECRET_ACCESS_KEY=<your_aws_access_key>
heroku config:set DJANGO_AWS_STORAGE_BUCKET_NAME=s3.foobar.com
heroku config:set MAILGUN_DOMAIN=mg.foobar.com
heroku config:set MAILGUN_API_KEY=key-<your_mailgun_key>
heroku config:set MAILGUN_SENDER_DOMAIN=mg.foobar.com
heroku config:set SENTRY_DSN=<your_sentry_dsn>
```

3.3.4 Deploy your project to heroku

After setting all those configuration variables, you should be able to deploy your project to heroku:

```
git push heroku master
```

And create a superuser for your production system:

```
heroku run python manage.py createsuperuser
```

Finally you should be able to check your deployment and open the website:

```
heroku run python manage.py check --deploy
heroku open
```

3.3.5 Use your own domain name with heroku

Just follow the instructions on the custom-domains help site at heroku.

3.3.6 SSL

Caution: This only works with paid heroku plans (hobby and upwards).

Install letsencrypt client

For paid dynos there's automatic certificate management with heroku-ssl available. If you are using a hobby dyno, you have to upload your certificates manually. A first step is to install certbot on your local machine.

```
brew install certbot
```

Prepare your app for verification

Add this to your “config/urls.py” file:

```
...
    from django.http import HttpResponse
    ...

urlpatterns = [
    ...
    # letsencrypt
    path(
        ".well-known/acme-challenge/{settings.LETSENCRYPT_VERIFICATION_URL}",
        letsencrypt_view,
        name="letsencrypt",
    ),
    ...
```

And this to your “config/settings/base.py” file:

```
# Letsencrypt
LETSENCRYPT_VERIFICATION_URL=env("LETSENCRYPT_VERIFICATION_URL")
LETSENCRYPT_VERIFICATION_DATA=env("LETSENCRYPT_VERIFICATION_DATA")
```

To verify your domain ownership you need to serve a snipped of data under a specific url. Both provided by letsencrypt if you run this command. Stop after you see the “Waiting for verification” message from certbot.

```
sudo certbot certonly --manual
```

Now you have to set those two letsencrypt environment variables. The cerbot client will show the content of those variables in the output:

```
heroku config:set LETSENCRYPT_VERIFICATION_URL=<the_part_after acme-challenge/>
heroku config:set LETSENCRYPT_VERIFICATION_DATA=<the_part_after file containting just␣
↪this data:>
git add .
git commit -m "added letsencrypt endpoint"
git push heroku master
```

You should check if you get the correct data from your site. If that’s the case you can now press <enter> on certbots verification step. If all went well, it will show you a congratulation message and tell you the location of the certificate.

You now need to add the certificate and key to heroku:

```
heroku certs:add /etc/letsencrypt/live/your_domain_name/fullchain.pem /etc/
↪letsencrypt/live/your_domain_name/privkey.pem
```

3.3.7 Caveats

Static Files

I couldn’t get serving static files to work with amazon S3. One problem was that DJANGO_AWS_STORAGE_BUCKET_NAME in the STATIC_URL setting seems to get ignored by the static templatetag resulting in a permanent redirect error page from S3. And the other problem is that S3 didn’t support https (broken certificate). But all static urls are https by default, so this didn’t work either. Maybe you can fix that by using a cloudfront distribution etc. but using whitebox to serve static files worked out of the box.

3.4 Setting up your production machine on EC2

3.4.1 Creating your machine on EC2

- Generate keypair for your machine
- Use the Ubuntu 16.04 image
- Add rules to security policy to allow ssh/http/https for inbound traffic
- Assign an elastic ip to your instance

3.4.2 Install required software to run your docker deployment

Switch to root user:

```
sudo su -
```

Update software on the image:

```
apt update && apt dist-upgrade
```

Install Docker, Supervisor and docker-compose to be able to run your production deployment automatically:

```
apt install python3 python3-pip docker.io supervisor docker-compose
```

3.4.3 Make supervisorctl accessible for normal users

Change the line chmod=0700 to chmod=0766 in /etc/supervisor/supervisord.conf

3.4.4 Make docker-compose runable as normal user

Add the default ubuntu EC2 user to /etc/group to make docker-compose executable by ubuntu.

```
usermod -a -G docker ubuntu
```

3.4.5 Reboot

Reboot the machine:

```
shutdown -r now
```

3.4.6 Check out source

Clone foobar repository into site directory:

```
git clone git@github.com:your_github_username/foobar.git site
```

3.4.7 Convenience

Add 'cd site' at the and of ~/.bashrc to automatically switch into the project directory on login.

3.4.8 Keeping the service running with supervisor

Create a link to supervisor.conf:

```
sudo su -
cd /etc/supervisor/conf.d/
ln -s /home/ubuntu/site/foobar.conf foobar.conf
/etc/init.d/supervisor restart
```

3.4.9 Set the environment variables

Use the env.example template to set the production environment variables in .env.

3.4.10 Starting the docker containers manually

Make sure the containers are build and the the database relations are created.

```
cd site
docker-compose -f production.yml build
docker-compose -f production.yml run django ./manage.py migrate
docker-compose -f production.yml up
```

3.4.11 Using supervisorctl

Check the service is now running via supervisorctl:

```
supervisorctl start foobar
supervisorctl status foobar
```

CHAPTER 4

Using a CDN (AWS S3 + Cloudfront)

When using a CDN, s3 with cloudfront for example, there are some settings to put in your production config which are not really obvious:

```
AWS_AUTO_CREATE_BUCKET = True
AWS_S3_REGION_NAME = 'eu-central-1'  # if your region differs from default
AWS_S3_SIGNATURE_VERSION = 's3v4'
AWS_S3_FILE_OVERWRITE = True
AWS_S3_CUSTOM_DOMAIN = env('CLOUDFRONT_DOMAIN')
```

Took me some time to figure out these settings. Those are additional settings, assumed you already used the django-cookiecutter template.

CHAPTER 5

Analytics

There only very limited analytics support here. But it’s possible to import your webservers access.log.

5.1 Immport your webserver logfile

Depends on the format of your logfile. The only supported format at the moment is caddy.

5.1.1 Create a virtualenv on your production server

At first, create a analytics user via the django admin interface. I gave it the username ‘analytics’ for convenience. Then create a python virtualenv to run the collect analytics cronjob. It will read the caddy access.log and write the requests into to your django-cast Requests model via a rest-API. You’ll also need pandas in this environment, because the cleanup of request is done in pandas.

```
apt install virtualenvwrapper  # if you don't already have this installed
apt install libpq-dev  # maybe you'll need that for psycopg2 compilation..
mkvirtualenv -p /usr/bin/python3 your_app_name
pip install pandas
```

Create a local file with all the required environment variables you need to run django management commands locally - I named it ‘.analytics_env’:

```
USE_DOCKER=no
DJANGO_AWS_ACCESS_KEY_ID=
DJANGO_AWS_SECRET_ACCESS_KEY=
DJANGO_AWS_STORAGE_BUCKET_NAME=
DJANGO_SETTINGS_MODULE=config.settings.local
USERNAME=analytics
OBTAIN_TOKEN_URL=https://your_domain_name.com/api/api-token-auth/
```

Be sure that you are now able to run django management commands:

```
env $(cat .analytics_env | xargs) ./manage.py
```

5.1.2 Get the api token for your analytics user

If you provide the right password for your analytics user, you should now be able to retrieve the api token for that user.

```
env $(cat .analytics_env | xargs) ./manage.py get_api_token
```

Don't forget to add the api token to your '.analytics_env':

```
API_TOKEN=d387ca7e5d2bf4932f1e9e9c9c4caec808571b39
```

You'll need to add two additional environment variables to your '.analytics_env':

```
REQUEST_API_URL=https://your_domain_name.com/api/request/
ACCESS_LOG_PATH=/var/log/caddy/your_domain_name.access.log
```

5.1.3 Set up a cronjob to run every hour or so to import your logfile

At first, place a shell script named 'analytics_cron.sh' in your project dir that you want to execute as a cronjob. It might look like this:

```
#!/bin/bash

cd $HOME/your_project_dir
(env $(cat .analytics_env | xargs) $HOME/.virtualenvs/your_env_name/bin/python manage.
↪py access_log_import 2>&1) > access_log_import.log
```

Make this script executable:

```
chmod +x analytics_cron.sh
```

And finally create a cronjob running every hour or something like this:

```
crontab -e
0 * * * * cd $HOME/your_project_dir && ./analytics_cron.sh
```

CHAPTER 6

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/ephes/django-cast/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

6.1.4 Write Documentation

Django Cast could always use more documentation, whether as part of the official Django Cast docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/ephes/django-cast/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up *django-cast* for local development.

1. Fork the *django-cast* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:ephes/django-cast.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv cast
   $ workon cast
   $ cd django-cast/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 cast tests
   $ python runtests.py test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/ephes/django-cast/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ python runtests.py tests -k test_get_post_detail
```

CHAPTER 7

Credits

7.1 Development Lead

- Jochen Wersdörfer <jochen-djangocast@wersdoerfer.de>

7.2 Contributors

- Dominik Geldmacher <oryon@soila.de>

CHAPTER 8

History

8.1 0.1.29 (2020-01-03)

- Use poetry instead of requirements.txt and setup.py

8.2 0.1.28 (2019-06-03)

- Added some analytics support: import your access.log and view dashboard with hits/day,week
- Fixed pub_date bug, leading to safari not being able to update posts + some tests
- Use local web-player and subscribe button (didn't improve performance, though :( )
- Fixed detail content not included in feed (shownotes were missing) bug
- Added some deployment documentation for heroku, ec2 and docker
- Overwritable block for detail link in post list template + documentation

8.3 0.1.27 (2019-05-27)

- Extended documentation
- It's now possible to mark content as "for post detail page" only
- Changed documentation to work with comments
- Fixed comments dependencies in setup.py

8.4 0.1.26 (2019-05-23)

- Bugfix: i18n should now work, finally!!1 duh

8.5 0.1.25 (2019-05-23)

- Bugfix: i18n should now work, finally
- Bugfix: Allow empty chaptermarks text field + test

8.6 0.1.24 (2019-05-22)

- Use blog.email as itunes:email instead of blog.user.email
- Added author field to have user editable author name
- Translation should now work since locale dir is included in MANIFEST.in
- Include documentation in package
- Use visible_date as pubDate for feed and sort feed by -visible_date instead of -pub_date

8.7 0.1.23 (2019-05-16)

- Comment en/disabling per site/blog/post
- Fix duration extraction and small issues with the installation docs @jnns
- Support for comments by @oryon-dominik

8.8 0.1.22 (2019-04-28)

- Use proper time field for chaptermark start instead of char
- Improved test coverage
- Improved video dimension handling for handbrake generated portrait videos

8.9 0.1.21 (2019-04-24)

- Fixed package dependencies
- Better release docs

8.10 0.1.20 (2019-04-24)

- Fixed version history
- Better release docs

8.11 0.1.19 (2019-04-24)

- Added fulltext search
- Added filtering by date + some faceted navigation support
- use overwritable template block for feeds section (could be used for podlove subscribe button)

8.12 0.1.18 (2019-04-18)

- Fixed broken update view due to empty chaptermarks + test
- Fixed two image/video javascript bugs

8.13 0.1.17 (2019-04-15)

- Added chaptermarks feature
- Duration is now displayed correctly in podlove player
- If an audio upload succeeded, add the uploaded element to podcast audio select form

8.14 0.1.16 (2019-03-23)

- Finally, rtfd is working again, including screencast

8.15 0.1.15 (2019-03-23)

- Trying again… rtfd still failing

8.16 0.1.14 (2019-03-23)

- Added rtfd configuration file to be able to use python 3 :/

8.17 0.1.13 (2019-03-22)

- Release to update read the docs

8.18 0.1.12 (2019-03-22)

- Improved installation documentation

8.19 0.1.11 (2019-03-21)

- Fixed requirements for package

8.20 0.1.10 (2019-03-21)

- Dont limit the number of items in feed (was 5 items)
- Workaround for ogg files (ending differs for Audio model field name)
- Added opus format to Audio model

8.21 0.1.9 (2019-03-12)

- Added some podcast specific fields to post edit form
- If two audio uploads have the same name, add them to the same model instance
- Added audio file support for post edit form
- Show which audio files already were uploaded

8.22 0.1.8 (2019-02-28)

- Added support for m4v and improved dimension detection for iOS videos
- Added some tests for different video sources

8.23 0.1.7 (2019-02-28)

- forgot linting

8.24 0.1.6 (2019-02-28)

- Use filepond for media uploads (images video)
- Improved portrait video support
- Get api prefix programatically from schema
- Fixed link to podcast in itunes (was feed, now it's post list)
- Set visible date to now if it's not set
- use load static instead of staticfiles (deprecated)
- Fixed language displayed in itunes (you have to set it in base.py in settings)
- Dont try to be fancy, just display a plain list of feed on top of post list site (and podcast feeds only if blog.is_podcast is True)

8.25 0.1.5 (2018-11-21)

- basic feed support (rss/atom) for podcasts
- travis now runs tests with ffprobe, too
- documentation fixes from @SmartC2016 and @oryon-dominik

8.26 0.1.4 (2018-11-18)

- Include css via cast_base.html
- audio fixes

8.27 0.1.3 (2018-11-17)

- Fixed css/static icons
- Merged pull request from SmartC2016 to fix javascript block issue
- Added some documentation

8.28 0.1.2 (2018-11-08)

- Added some requirements
- Release Documentation

8.29 0.1.1 (2018-11-07)

- Travis build is ok.

8.30 0.1.0 (2018-11-05)

- First release on PyPI.