CHIRP Radio Documentation

Release 1.0

Kumar McMillan and contributors

April 18, 2016

Contents

1 Documentation 3

2 The CHIRP API 9

3 Community 11

4 Other code 13

5 Indices and tables 15

CHIRP is a non-profit organization that runs a community radio station in Chicago focused on new music and the arts. This is the source code for some internal applications we are building.

- Listen to CHIRP Radio here
- Keep up with the CHIRP organization here

These are internal tools used in the daily operation of our station. In particular, this is not meant to be a turnkey system for running a radio station. Many things about this code are CHIRP-specific. However, all code is open source and we may be able to provide guidance to anyone wishing to generalize the apps and modules for other uses, just ask.

We use Google App Engine and Django as the platform for these applications.

CHAPTER 1

Documentation

Here's how to get started developing CHIRP Radio DJ applications.

1.1 Installing

- *Using Git*
- *Prerequisites*
- *Running The Development Server*
- *Running The Test Suite*

1.1.1 Using Git

To create a new local repository go to https://github.com/chirpradio/chirpradio and fork the repository to your own *username* account. Check out your clone at a URL like this:

```
git clone git@github.com:username/chirpradio.git
```

You can use your local fork to create topic branches and make pull requests into the main repo. Here is a guide on working with topic branches.

1.1.2 Prerequisites

Everything should run in Python 2.5 or greater http://python.org/download/

Note: Recent Ubuntu Linux versions (at least after Jaunty) ship with Python 2.6. Many have reported problems running the Google App Engine SDK with a non-2.5.* version of Python. To install Python 2.5 without breaking the default Python install, you can use this command:

```
sudo apt-get install python2.5
```

Install the Google App Engine SDK from http://code.google.com/appengine/downloads.html

If on Mac OS X be sure to start up the launcher once so that it prompts you to create symbolic links in /usr/local/google_appengine

Unlike the Google App Engine Python SDK for Mac OS X/Windows, the Linux version comes as a zip archive rather than an installer. To install, just unpack the archive into /usr/local/google_appengine. Or you can unpack it to your home directory and create a symlink in /usr/local/google_appengine.

It's a good idea to install PyCrypto for pushing code to Google and so that the SDK works as expected.

On a Debian/Ubuntu system, use this command:

```
sudo apt-get install python-crypto
```

On Mac OS X you need to grab the PyCrypto source and run:

```
sudo python setup.py install
```

To run the JavaScript lint tests (which will fail otherwise) you will need the jsl command line tool, aka javascript-lint.

On a Mac OS X system *with* homebrew, type:

```
brew install jsl
```

(there is probably something similar for Linux)

1.1.3 Running The Development Server

Note: The Google App Engine SDK currently does not run inside a virtualenv. This is a known bug.

To start up a local server, run

```
python manage.py runserver
```

Note: If you are running on a system with multiple versions of Python installed, make sure that you are using the 2.5 version, e.g.:

```
python2.5 manage.py runserver
```

You can reach your local server by going to http://localhost:8000/ in your web browser.

If you are running this server on a different computer, you need to run the server with

```
python manage.py runserver 0.0.0.0
```

instead. This tells Django to bind to your external IP address and accept remote connections.

Below, we refer to local URLs like this: http://HOST:PORT/some/url You should replace "HOST:PORT" with the appropriate host name/port combination.

1.1.4 Running The Test Suite

To run all unit tests:

```
python manage.py test
```

You can also use

```
python manage.py test [application name]
```

to only run a single application's tests.

1.2 Bootstrapping Your Dev Setup

To do anything interesting with your development site you'll need some test data and other bits of scaffolding.

1.2.1 Creating a New Local Test User

If you are running locally, you can create a test account by:

1. Go to http://HOST:PORT/_ah/login
2. Enter the email address that you want to use for testing, and check the "sign in as administrator" box. Then click the "login" button.
3. Go to http://HOST:PORT/auth/_bootstrap. Hitting this URL will create a new user account and then immediately redirect you to a login page.
4. Now log in using the email address that you chose in step 2 and the password "test".

The test user created by this method has superuser privileges, so you should be able to add other test accounts by visiting http://HOST:PORT/auth/

1.2.2 Resetting a Local Test User's Password

Since a local development instance cannot send email, the normal password recovery mechanism cannot be used for test accounts. If you forget a test account's password, you can

1. Go to http://HOST:PORT/_ah/admin/datastore?kind=User
2. Find the user whose password you wish to reset, then click on the "Key" hyperlink in order to edit it.
3. Replace the entity's password attribute with the following: 32e6e8b1d913ca40bd3f1d683ba65925bba1f559381f
4. Click the "Save Changes" button.

You should now be able to log in as that user with the password "test".

1.2.3 Working With Artists and Albums

To see some artists and albums in the DJ Database, open this special URL: http://127.0.0.1:8000/djdb/_bootstrap

This adds David Bowie, The Clash, and a few others.

1.2.4 Working With the Datastore Config

Open http://127.0.0.1:8000/common/_init_config to initialize the Datastore config object.

1.3 Creating a Test Instance on App Engine

At some point, it is a good idea to test your code changes in a real App Engine environment. You may also need a test instance in order to test out the features of chirpradio/chirpradio-machine. Below you will find instructions for deploying and configuring a test instance that is completely separate from the production site.

1. Go to the App Engine console and, from the top menu, select "Create project".
2. Name the project chirpradio-test and hit Create.
3. Note the project ID. It will be something like chirpradio-test-123.
4. Go to the directory where you cloned chirpradio/chirpradio.
5. To upload the code to the new project, run this at the command line:

```
appcfg.py -A <your project id> update .
```

6. Go to `http://<your project id>.appspot.com` and verify that the site is up.
7. Go to the API Manager credentials page.
8. Select Create credentials > Service account key.
9. Select App Engine default service account, choose JSON, and hit Create. The newly-generated key will be automatically downloaded to your computer.
10. Connect to your remote datastore using the Remote API Shell:

```
GOOGLE_APPLICATION_CREDENTIALS=/path/to/service_account_key.json remote_api_shell.py -s <your pr
```

11. To create a new user, run the following code in the shell:

```
from auth.models import User
user = User(email='first.last@email.com', first_name='First', last_name='Last', is_superuser=Tru
user.set_password('password')
user.save()
```

12. Go ahead and login using the email and password you set for your user in the Remote API Shell.

1.4 Users and Authentication

The chirpradio applications use custom middleware to enforce access controls. It will automatically take care of details like blocking inactive users or redirecting unauthenticated users to the login page.

1.4.1 Roles

Roles are a light-weight substitute for the standard Django auth module's notion of groups.

The list of valid roles is hard-wired into auth/roles.py, so adding a new role requires an updated version of the app to be pushed into production.

1.4.2 Access Policy

With only a very few exceptions, all of the URLs that are part of the chirpradio applications are only accessible to signed-in users. If an unauthenticated user tries to visit such a URL, they will be redirected to a login page, and then redirected back to the originally-requested page after they have successfully signed in. This behavior is controlled by custom middleware defined in auth/middleware.py.

Access can be further restricted based on role using the decorators defined in auth/decorators.py. For example, this is how to define a view that is only accessible to a user who has the role "volunteer coordinator":

```
from auth import roles
from auth.decorators import require_role

@require_role(roles.VOLUNTEER_COORDINATOR)
def my_view(request):
    ... etc ...
```

1.4.3 User Information

Our User object is defined in auth/models.py. It is similar, but not identical, to the stock Django User object.

For any incoming HttpRequest, the user attribute is automatically populated with the logged-in user's User object.

```
def my_hello_world_view(request):
    return HttpResponse('Hello %s!' % request.user)
```

Users are keyed on their email addresses:

```
some_user = User.get_by_email(email_addr)
```

However, users are allowed to change their email address. Applications should not put them in the datastore or otherwise assume that they are invariant.

1.4.4 Unit Testing

To simplify unit testing, the CHIRP authentication system is integrated with Django's django.test.client module. You can use the login method to test against fake users with various characteristics.

```
from django.test.client import Client

my_client = Client()
# You can set any of the User object's attributes here.
my_client.login(email="test@test.com", roles=[role1, role2])
response = my_client.get("/some/page/to/test")
```

For more information on unit testing in Django, please see http://docs.djangoproject.com/en/1.0/topics/testing/

1.5 Adding a New Application

Every application has a name that looks like this: "landing_page". Your code lives in a directory with the same name. Your templates go under the directory templates/[application name]. Your media files go under the directory media/[application name].

All of your URLs are automatically mapped to be under http://HOST:PORT/appname/my/url

To make your URLs visible, you need to:

1. Update the top-level urls.py to include your urls.
2. Add your application to INSTALLED_APPS in settings.py.

1.6 The CHIRP Radio Style Guide

1.6.1 Coding Conventions

In general, Python code should follow the guidelines and conventions outlined in PEP 8.

A few additional rules:

- One-character variable names are strongly discouraged, except for variables of iteration.

- Avoid "power features" like metaclasses, import hacks, reflection, etc. These features are occasionally necessary for low-level hacks in core infrastructure, but should generally not occur in applications. Simplicity in code is a virtue.
- Code should always be accompanied by unit tests.
- Always use new-style classes by deriving from object in base classes.

1.6.2 Imports

Long lists of imports can be confusing and difficult to scan and maintain. To avoid this, the encouraged order to imports is.

- First standard Python modules.
- Then core Django modules.
- Then Google App Engine-specific modules.
- The core chirpradio infrastructure.
- Finally, your application or subcomponent.

Each group of imports should occur in alphabetical order.

1.6.3 Copyright & License Notice

When you create a new source file, please include this notice at the top:

```
###
### Copyright [CURRENT YEAR] The Chicago Independent Radio Project
### All Rights Reserved.
###
### Licensed under the Apache License, Version 2.0 (the "License");
### you may not use this file except in compliance with the License.
### You may obtain a copy of the License at
###
###     http://www.apache.org/licenses/LICENSE-2.0
###
### Unless required by applicable law or agreed to in writing, software
### distributed under the License is distributed on an "AS IS" BASIS,
### WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
### See the License for the specific language governing permissions and
### limitations under the License.
###
```

If you edit a file whose copyright year is in the past, do not replace it with the current year. The year in a file should reflect the year that the file was created, not the year it was last edited.

1.6.4 TODO Comments

In the code, you will occasionally see comments of the form `# TODO(username): Need to do so-and-so in the future.`

The "username" indicates the person who originally made the note, *not* the person who is assigned to fix it. The name is there so that you can know who to ask if you have questions about the note.

If you are new to the project, a good way to get started is to search for TODO items and try to do them.

CHAPTER 2

The CHIRP API

There is a hosted API to get CHIRP data for mobile apps.

CHAPTER 3

Community

Feel free to connect and ask questions on the CHIRP Dev mailing list.

CHAPTER 4

Other code

Looking for something else by CHIRP?

- The CHIRP iPhone app
- The CHIRP Android app
- The CHIRP Radio Machine: music library / broadcast stream

CHAPTER 5

Indices and tables

- genindex
- modindex
- search