Shiva Documentation

Release 0.9.0

Alvaro Mouriño

March 02, 2015

Contents:

CHAPTER 1

Intro

Shiva is, technically speaking, a RESTful API to your music collection. It indexes your music and exposes an API with the metadata of your files so you can then perform queries on it and organize it as you wish.

On a higher level, however, Shiva aims to be a free (as in freedom and beer) alternative to popular music services. It was born with the goal of giving back the control over their music and privacy to the users, protecting them from the industry's obsession with control.

It's not intended to compete directly with online music services, but to be an alternative that you can install and modify to your needs. You will own the music in your server. Nobody but you (or whoever you give permission) will be able to delete files or modify the files' metadata to correct it when it's wrong. And of course, it will all be available to any device with Internet connection.

You will also have a clean, RESTful API to your music without restrictions. You can grant access to your friends and let them use the service or, if they have their own Shiva instances, let both servers talk to each other and share the music transparently.

To sum up, Shiva is a distributed social network for sharing music.

Read more on https://hacks.mozilla.org/2013/03/shiva-more-than-a-restful-api-to-your-music-collection/

CHAPTER 2

Installation

Before installing shiva there are some dependencies that you have to take care of.

2.1 Prerequisites

You are going to need:

- ffmpeg
- libxml C headers
- python headers
- sqlite (optional)

If you want Shiva to automatically fetch artists' images from Last.FM while indexing you are going to need an API key. You can get one at http://www.last.fm/api/account/create

This makes the whole indexing slower because issues a request on a per-album and per-artist basis, but does a lot of work automatically for you.

By default Shiva uses a SQLite database, but this can be overriden.

To install all the dependencies on Debian (and derivatives):

```
sudo apt-get install libxml2-dev libxslt-dev ffmpeg python-dev sqlite
```

For simplicity, you can just run:

```
sudo apt-get install `cat dependencies.apt`
```

If at some point of the installation process you get the error:

```
/usr/bin/ld: cannot find -lz
```

You also need the package `lib32z1-dev`

On Mac OS X with homebrew you can get the libxml headers with:

```
brew install libxml2 libxslt
```

On Mac OS X sqlite should come pre-installed. If it's not:

```
brew install sqlite
```

2.2 Installing from source

- Get the source:

```
$ git clone https://github.com/tooxie/shiva-server.git
$ cd shiva-server
```

- Create and activate your virtalenv (highly recommended):

```
$ virtualenv .virtualenv
$ source .virtualenv/bin/activate
```

- Install:

```
$ python setup.py develop
```

- Rename shiva/config/local.py.example to local.py:

```
$ cp shiva/config/local.py.example shiva/config/local.py
```

See **'Configuration'_** for more info.

- Edit it and configure the directories to scan for music.
  - See **'Scanning directories'_** for more info.
- Run the indexer:

```
$ shiva-indexer
```

- Run the file server:

```
$ shiva-fileserver
```

- Run the server in a different console:

```
$ shiva-server
```

- Point your browser to a Resource, like: http://127.0.0.1:9002/artists (See **'Base Resources'_**)

2.3 Installing using pip

You can install Shiva through `pip`, running the following command:

```
$ pip install shiva
```

That will automatically download and install Shiva and all its dependencies.

Note: This will install the latest release, which may contain bugs and lack some features. It is highly recommended that you install the latest development version, following the manual installation guide above.

CHAPTER 3

Configuration

Shiva looks for config files in the following places:

- `config/local.py` relative to the directory where Shiva is installed.
- If an environment variable `$SHIVA_CONFIG` is set, then is assumed to be pointing to a config file.
- `$XDG_CONFIG_HOME/shiva/config.py` which defauls to `$HOME/.config/shiva/config.py`.

If all 3 files exist, then all 3 will be read and overriden in that same order, so `$XDG_CONFIG_HOME/shiva/config.py` will take precedence over `config/local.py`.

3.1 SECRET_KEY

It's mandatory that you define a `SECRET_KEY` in your local configuration file, `shiva-server` won't start otherwise. However, shiva will suggest you one which will be based on all printable characters, except for spaces and quotes. Read the source of `shiva.utils.randstr` for more information.

The key will be used to sign the authentication tokens, so make sure that it's long, random, and securely generated. Don't ever use any 3rd party service for this.

3.2 DEBUG mode

It's possible to load settings specific for debugging. If you have the following in any of your config files:

```
DEBUG = True
```

Then Shiva will also try to load this configuration files:

- `config/debug.py` relative to the directory where Shiva is installed.
- `$XDG_CONFIG_HOME/shiva/debug.py` which defauls to `$HOME/.config/shiva/debug.py`.

In this case `$XDG_CONFIG_HOME/shiva/debug.py` will also have precedence over `config/debug.py`.

CHAPTER 4

Indexing your music

The indexer receives the following command line arguments.

- `--lastfm`
- `--hash`
- `--nometadata`
- `--reindex`
- `--write-every=<num>`

If you set the `--lastfm` flag Shiva will retrieve artist and album images from Last.FM, but for this to work you need to get an API key (see **'Prerequisites'_**) and include it in your `local.py` config file.

When `--hash` is present, Shiva will hash every file using the md5 algorithm, in order to find duplicates, which will be ignored. Note that this will decrease indexing speed notably.

The `--nometadata` option saves dummy tracks with only path information, ignoring the file's metadata. This means that albums and artists will not be saved, but indexing will be as fast as it gets.

If both the `--nometadata` and `--lastfm` flags are set, `--nometadata` will take precedence and `--lastfm` will be ignored.

With `--reindex` the whole database will be dropped and recreated. Be careful, all existing information **will be deleted**. If you just want to update your music collection, run the indexer again **without** the `--reindex` option.

The indexer is optimized for performance; hard drive hits, like file reading or DB queries, are done as few as possible. As a consequence, memory usage is quite heavy. Keep that in mind when indexing large collections.

To keep memory usage down, you can use the `--write-every` parameter. It receives a number and will write down to disk and clear cache after that many tracks indexed. If you pass 1, it will completely ignore cache and just write every track to disk. This has the lowest possible memory usage, but as a downside, indexing will be much slower.

It's up to you to find a good balance between the size of your music collection and the available RAM that you have.

4.1 Restricting extensions

If you want to limit the extensions of the files to index, just add the following config to your `local.py` file:

```
ALLOWED_FILE_EXTENSIONS = ('mp3', 'ogg')
```

That way only 'mp3' and 'ogg' files will be indexed.

4.2 Scanning directories

To tell Shiva which directories to scan for music, you will have to configure your `shiva/config/local.py` file. There you will find a `MEDIA_DIRS` option where you need to supply `MediaDir` objects.

This object allows for media configuration. By instantiating a `MediaDir` class you can tell Shiva where to look for the media files and how to serve those files. It's possible to configure the system to look for files on a directory and serve those files through a different server.

```
MediaDir(root='/srv/http', dirs=('/music', '/songs'),
         url='http://localhost:8080/')
```

Given that configuration Shiva will scan the directories `/srv/http/music` and `/srv/http/songs` for media files, but they will be served through `http://localhost:8080/music/` and `http://localhost:8080/songs/`.

If just a dir is provided you will also need to run the file server, as mentioned in the installation guide. This is a simple file server, for testing purposes only. Do **NOT** use in a live environment.

```
MediaDir('/home/fatmike/music')
```

For more information, check the source of *shiva/media.py*.

CHAPTER 5

Shiva client

The Shiva-Client is a web-based front-end for Shiva-Server built as a single page application using HTML5 technologies. It includes its own test web server so you don't need to install one.

5.1 Wish you were here

Or you can also build your own client and put your own ideas into practice. I encourage you to do so. Build your own music player that meets your exact needs.

Feel free to issue a PR if you need new functionality in Shiva.

CHAPTER 6

Base Resources

You have the following resources available:

- `/artists`
- `/artists/<int:artist_id>`
- `/artists/<int:artist_id>/shows`
- `/albums`
- `/albums/<int:album_id>`
- `/tracks`
- `/tracks/<int:track_id>`
- `/tracks/<int:track_id>/lyrics`

And some meta resources:

- `/random/<str:resource_name>`
- `/whatsnew`
- `/clients`
- `/about`

6.1 /artists

Example response for the request `GET /artists/3`:

```
{
    "name": "Eterna Inocencia",
    "image": "http://userserve-ak.last.fm/serve/_/8339787/Eterna+Inocencia+Eterna.jpg",
    "uri": "/artists/3",
    "slug": "eterna-inocencia",
    "id": 3
}
```

6.1.1 Fields

- `id`: The object's ID.

- image: Link to a photo. (Provided by last.fm)
- name: The artist's name.
- slug: A slug of the artist's name.
- uri: The URI of this resource's instance.

6.1.2 /artists/<int:id>/shows

Information provided by BandsInTown. This is the only resource that is not cached in the local database given to it's dynamic nature.

Example response for the request `GET /artists/1/shows`:

```
[
    {
        "other_artists": [
            {
                "mbid": "5c210861-2ce2-4be3-9307-bbcfc361cc01",
                "facebook_tour_dates_url": "http://bnds.in/kVwY1Y",
                "image_url": "http://www.bandsintown.com/Pennywise/photo/medium.jpg",
                "name": "Pennywise",
            }
        ],
        "artists": [
            {
                "id": 1,
                "uri": "/artists/1"
            }
        ],
        "tickets_left": true,
        "title": "Lagwagon @ Commodore Ballroom in Vancouver, Canada",
        "venue": {
            "latitude": "49.2805760",
            "name": "Commodore Ballroom",
            "longitude": "-123.1207430"
        },
        "id": "6041814",
        "datetime": "Thu, 21 Feb 2013 19:00:00 -0000"

    }
]
```

6.1.3 Fields

- other_artists: A list with artists that are not in Shiva's database.
  - mbid: MusicBrainz.org ID.
  - facebook_tour_dates_url: URI to BandsInTown's Facebook app for this artist.
  - image_url: URI to an image of the artist.
  - name: Name of the artist.
- artists: A list of artist resources.

- `tickets_left`: A boolean representing the availability (or not) of tickets for the concert.
- `title`: The title of the event.
- `venue`: A structure identifying the venue where the event takes place.
  - `latitude`: Venue's latitude.
  - `name`: Venue's name.
  - `longitude`: Venue's longitude.
- `id`: BandsInTown's ID for this event.
- `datetime`: String representation of the date and time of the show.

6.1.4 Parameters

The Shows resource accepts, optionally, two pairs of parameters:

- `latitude` and `longitude`
- `country` and `city`

By providing one of this two pairs you can filter down the result list only to a city. If only one of the pair is provided (e.g., only city) will be ignored, and if both pairs are provided, the coordinates will take precedence.

6.2 /albums

Example response for the request `GET /albums/9`:

```
{
    "artists": [
        {
            "id": 2,
            "uri": "/artists/2"
        },
        {
            "id": 5,
            "uri": "/artists/5"
        }
    ],
    "name": "NOFX & Rancid - BYO Split Series (Vol. III)",
    "year": 2002,
    "uri": "/albums/9",
    "cover": "http://userserve-ak.last.fm/serve/300x300/72986694.jpg",
    "id": 9,
    "slug": "nofx-rancid-byo-split-series-vol-iii"
}
```

6.2.1 Fields

- `artists`: A list of the artists involved in that record.
- `cover`: A link to an image of the album's cover. (Provided by last.fm)
- `id`: The object's ID.
- `name`: The album's name.

- `slug`: A slug of the album's name.
- `uri`: The URI of this resource's instance.
- `year`: The release year of the album.

6.2.2 Filtering

The album list accepts an `artist` parameter in which case will filter the list of albums only to those corresponding to that artist.

Example response for the request `GET /albums?artist=7`:

```
[
    {
        "artists": [
            {
                "id": 7,
                "uri": "/artists/7"
            }
        ],
        "name": "Anesthesia",
        "year": 1995,
        "uri": "/albums/12",
        "cover": "http://userserve-ak.last.fm/serve/300x300/3489534.jpg",
        "id": 12,
        "slug": "anesthesia"
    },
    {
        "artists": [
            {
                "id": 7,
                "uri": "/artists/7"
            }
        ],
        "name": "Kum Kum",
        "year": 1996,
        "uri": "/albums/27",
        "cover": "http://userserve-ak.last.fm/serve/300x300/62372889.jpg",
        "id": 27,
        "slug": "kum-kum"
    }
]
```

Giving '0' instead you get the albums with no artist. If the argument is non-numeric you will get a `400 Bad Request` error.

6.3 /tracks

Example response for the request `GET /tracks/510`:

```
{

    "ordinal": 4,
    "bitrate": 128,
    "slug": "dinosaurs-will-die",
    "album": {
        "id": 35,
        "uri": "/albums/35"
    },
    "title": "Dinosaurs Will Die",
    "artist": {
        "id": 2,
        "uri": "/artists/2"
    },
    "uri": "/tracks/510",
    "id": 510,
    "length": 180,
    "files": {
        "audio/mp3": "http://localhost:8080/nofx-pump_up_the_valuum/04. Dinosaurs Will Die.mp3",
        "audio/ogg": "/tracks/510/convert?mimetype=audio%2Fogg"
    }

}
```

6.3.1 Fields

- `album`: The album to which this track belongs.
- `bitrate`: In MP3s this value is directly proportional to the sound quality.
- `id`: The object's ID.
- `length`: The length in seconds of the track.
- `ordinal`: The ordinal of this track with respect to this album.
- `slug`: A slug of the track's title.
- `title`: The title of the track.
- `uri`: The URI of this resource's instance.
- `files`: A list of URIs to access the files in the different formats, according to the MEDIA_DIRS setting.

6.3.2 Filtering by artist

Example response for the request `GET /tracks?artist=16`:

```
[
    {
        "ordinal": 1,
        "bitrate": 196,
        "slug": "pay-cheque-heritage-ii",
        "album": {
            "id": 36,
            "uri": "/albums/36"
        },
        "title": "Pay Cheque (Heritage II)",
        "artist": {
            "id": 16,
            "uri": "/artists/16"
        },
        "uri": "/tracks/523",
        "id": 523,
        "length": 189,
        "files": {
            "audio/mp3": "http://localhost:8080/ftd-2003-sofa_so_good/01 For The Day - Pay Cheque (He
            "audio/ogg": "/tracks/523/convert?mimetype=audio%2Fogg"
        }
    },
    {
        "ordinal": 2,
        "bitrate": 186,
        "slug": "in-your-dreams",
        "album": {
            "id": 36,
            "uri": "/albums/36"
        },
        "title": "In Your Dreams",
        "artist": {
            "id": 16,
            "uri": "/artists/16"
        },
        "uri": "/tracks/531",
        "id": 531,
        "length": 171,
        "files": {
            "audio/mp3": "http://localhost:8080/ftd-2003-sofa_so_good/02 For The Day - In Your Dreams
            "audio/ogg": "/tracks/523/convert?mimetype=audio%2Fogg"
        }
    }
]
```

Giving ‘0’ instead you get the tracks with no artist. If the argument is non-numeric you will get a `400 Bad Request` error.

6.3.3 Filtering by album

Example response for the request `GET /tracks?album=18`:

```
[

    {
        "album": {
            "id": 18,
            "uri": "/albums/18"
        },
        "length": 132,
        "files": {
            "audio/mp3": "http://localhost:8080/flip-keep_rockin/flip-01-shapes.mp3",
            "audio/ogg": "/tracks/277/convert?mimetype=audio%2Fogg"
        }
        "ordinal": 1,
        "title": "Shapes",
        "slug": "shapes",
        "artist": {
            "id": 9,
            "uri": "/artists/9"
        },
        "bitrate": 192,
        "id": 277,
        "uri": "/tracks/277"
    },
    {
        "album": {
            "id": 18,
            "uri": "/albums/18"
        },
        "length": 118,
        "files": {
            "audio/mp3": "http://localhost:8080/flip-keep_rockin/flip-02-stucked_to_the_ground.mp3",
            "audio/ogg": "/tracks/281/convert?mimetype=audio%2Fogg"
        }
        "ordinal": 2,
        "title": "Stucked to The Ground",
        "slug": "stucked-to-the-ground",
        "artist": {
            "id": 9,
            "uri": "/artists/9"
        },
        "bitrate": 192,
        "id": 281,
        "uri": "/tracks/281"
    }
]
```

Giving '0' instead you get the tracks with no album. If the argument is non-numeric you will get a `400 Bad Request` error.

6.3.4 Listing orphan tracks

It is possible to obtain the list of tracks with no album by giving '0' to the `album` parameter. The same is true for the `artist` parameter. By combining both (`/tracks?album=0&artist=0`) you can get the list of "orphan" tracks.

6.4 Track creation

To create a track, `POST` a Multipart-Encoded file as a `track` argument to the `/tracks` resource. This is the only required parameter. If you include the arguments `artist_id` or `album_id`, it will take precedence and the file's metadata for artist (or album) will be ignored. If you send multimple IDs, they will all be used, but if any of them doesn't exist in the DB, the system will return a `400 Bad Request` and the track won't be saved.

```
curl -F "track=@file.mp3" -F "artist_id=1" -F "artist_id=17" http://127.0.0.1:9002/tracks
```

Use the query arguments `hash_file` and `no_metadata` to define if the file gets hashed and its metadata read.

```
curl -F "track=@file.mp3" http://127.0.0.1:9002/tracks?hash_file=true&no_metadata=true
```

6.5 /tracks/<int:id>/lyrics

Example response for the request `GET /tracks/256/lyrics`:

```
{
    "track": {
        "id": 256,
        "uri": "/tracks/256"
    },
    "source_uri": "http://lyrics.com/eterna-inocencia/my-family/",
    "id": 6,
    "uri": "/lyrics/6"
}
```

6.5.1 Fields

- `id`: The object's ID.
- `source_uri`: The URI where the lyrics were fetched from.
- `track`: The track for which the lyrics are.
- `uri`: The URI of this resource's instance.

6.5.2 Adding more lyric sources

Everytime you request a lyric, Shiva checks if there's a lyric associated with that track in the database. If it's there it will immediately retrieve it, otherwise will iterate over a list of scrapers, asking each one of them if they can fetch it. This list is in your local config file and looks like this:

```
SCRAPERS = {
    'lyrics': (
        'modulename.ClassName',
    ),
}
```

This will look for a class `ClassName` in `shiva/lyrics/modulename.py`. If more scrapers are added, each one of them is called sequentially, until one of them finds the lyrics and the rest are not executed.

6.5.3 Adding scrapers

If you want to add your own scraper just create a file under the lyrics directory, let's say `mylyrics.py` with this structure:

```python
from shiva.lyrics import LyricScraper

class MyLyricsScraper(LyricScraper):
    """ Fetches lyrics from mylyrics.com """

    def fetch(self, artist, title):
        # Magic happens here

        if not lyrics:
            return False

        self.lyrics = lyrics
        self.source = lyrics_url

        return True
```

And then add it to the scrapers list:

```
SCRAPERS = {
    'lyrics': (
        'modulename.ClassName',
        'mylyrics.MyLyricsScraper',
    ),
}
```

Remember that the `fetch()` method has to return `True` in case the lyrics were found or `False` otherwise. It must also store the URL where they were fetched from in `self.source`. That's where Shiva looks for the information.

Shiva will **not** store the actual lyrics, only the URI where the lyric was found.

For more details check the source of the other scrapers.

6.6 The `fulltree` modifier

The three main resources accept a `fulltree` parameter when retrieving an instance. Those are:

- `/artists/<int:artist_id>`
- `/albums/<int:album_id>`
- `/tracks`
- `/tracks/<int:track_id>`

Whenever you set `fulltree` to any value that evaluates to `True` (i.e., any string except `'false'` and `'0'`) Shiva will include not only the information of the object you are requesting, but also the child objects.

Here's an example response for the request `GET /artists/2?fulltree=true`:

```
{
    "name": "Eterna Inocencia",
    "image": "http://userserve-ak.last.fm/serve/_/8339787/Eterna+Inocencia+Eterna.jpg",
    "uri": "/artists/2",
    "events_uri": null,
    "id": 2,
    "slug": "eterna-inocencia",
    "albums": [
        {
            "artists": [
                {
                    "id": 2,
                    "uri": "/artists/2"
                }
            ],
            "name": "Tomalo Con Calma EP",
            "year": 2002,
            "uri": "/albums/2",
            "cover": "http://spe.fotolog.com/photo/30/54/51/alkoldinamita/1230537010699_f.jpg",
            "id": 2,
            "slug": "tomalo-con-calma-ep",
            "tracks": [
                {
                    "album": {
                        "id": 2,
                        "uri": "/albums/2"
                    },
                    "length": 161,
                    "files": {
                        "audio/mp3": "http://127.0.0.1:8001/eterna_inocencia/tomalo-con-calma.mp3",
                        "audio/ogg": "/tracks/27/convert?mimetype=audio%2Fogg"
                    }
                    "ordinal": 0,
                    "title": "02 - Rio Lujan",
                    "slug": "02-rio-lujan",
                    "artist": {
                        "id": 2,
                        "uri": "/artists/2"
                    },
                    "bitrate": 192,
                    "id": 27,
                    "uri": "/tracks/27"
                },
                {
                    "album": {
                        "id": 2,
                        "uri": "/albums/2"
                    },
                    "length": 262,
                    "files": {
                        "audio/mp3": "http://127.0.0.1:8001/eterna_inocencia/estoy-herido-en-mi-inter
                        "audio/ogg": "/tracks/28/convert?mimetype=audio%2Fogg"
                    }
                    "ordinal": 0,
                    "title": "03 - Estoy herido en mi interior",
                    "slug": "03-estoy-herido-en-mi-interior",
                    "artist": {
                        "id": 2,
                        "uri": "/artists/2"
                    },
                    "bitrate": 192,
                    "id": 28,
                    "uri": "/tracks/28"
                },
            ]
        }
    ]
}
```

6.6.1 Using `fulltree` on tracks

The behaviour on a track resource is a little different. In the previous example tracks are the leaves of the tree, but when the full tree of a track is requested then all the scraped resources are also included, like lyrics.

This is not the default behaviour to avoid DoS'ing scraped websites when fetching the complete discography of an artist.

Note that if you request the list of tracks with `fulltree`, only the related resources will be included (i.e.: artists and albums) but not the scraped ones.

6.6.2 Using `fulltree` on artists

The tree for artists will contain the extra field `no_album_tracks`, which is simply a list of tracks that are not related to any album:

```
no_album_tracks: [
    {
        album: null,
        artist: {
            id: 4,
            uri: "/artists/4"
        },
        bitrate: 192,
        files: {
            audio/mp3: "http://127.0.0.1:8001/music/dead_fish-1998-sirva-se/14-dead_fish-the_party-bu
            audio/ogg: "/tracks/82/convert?mimetype=audio%2Fogg"
        },
        id: 82,
        length: 1,
        ordinal: 0,
        slug: "14-dead-fish-the-party-buc",
        title: "14-dead fish-the party-buc",
        uri: "/tracks/82"
    }
],
```

6.7 Pagination

All the listings are not paginated by default. Whenever you request a list of either *artists*, *albums* or *tracks* the server will retrieve every possible result unless otherwise specified.

It is possible to paginate results by passing the `page_size` and the `page` parameters to the resource. They must both be present and be positive integers. If not, they will both be ignored and the whole set of elements will be retrieved.

An example request is `GET /artists?page_size=10&page=3`.

6.8 Using slugs instead of IDs

In previous versions of Shiva it was possible to use slugs instead of ID to request a specific resource. This is *not possible* anymore, since we use UUID values as IDs, and it's not possible to differentiate slugs from UUIDs without hitting the database every time. For performance reasons, slugs as record identifiers were discarded completely. However, slugs are still generated, stored, and present in the resulting JSON. Feel free to use them, but Shiva doesn't commit to keeping them unique.

6.8.1 Uniqueness of slugs

Slugs are *not* unique. Shiva does not commit to keeping slugs unique. For this reason, don't use them as identifiers.

CHAPTER 7

Meta Resources

Meta resources are simply dynamic or informational resources with no relation to any database model in particular. They are:

- `/random/<str:resource_name>`
- `/whatsnew`
- `/clients`
- `/about`

7.1 /random

You can request a random instance of a given resource for *artists*, *albums* or *tracks*. To do so you need to issue a GET request on one of the following resources:

- `/random/artist`
- `/random/album`
- `/random/track`

They all will return a consistent structure containing `id` and `uri`, as in this example response for the request `GET /random/artist`:

```
{
    "id": 3,
    "uri": "/artist/3"
}
```

You will have to issue another request to obtain the details of the instance.

7.2 /whatsnew

There's a special resource that lets you query the database to retrieve all the resources older than a given date, at the same time:

```
/whatsnew?since=YYYYMMDD
```

This will return an object with the following format:

```
{
    "artists": [],
    "albums": [
        {
            "id": 10,
            "uri": "/album/10"
        }
    ],
    "tracks": [
        {
            "id": 121,
            "uri": "/track/121"
        },
        {
            "id": 122,
            "uri": "/track/122"
        }
    ],
}
```

CHAPTER 8

Format conversion

No matter in which format files were indexed, it is possible to convert tracks to serve them in different formats. For this you are going to need `ffmpeg` installed in your system.

If you have `fmpeg` compiled but not installed, you can give Shiva the path to the binary in a setting, in this format:

```
FFMPEG_PATH = '/usr/bin/ffmpeg'
```

You will notice that track objects contain a `files` attribute:

```
{
    "id": 510,
    "uri": "/track/510",
    "files": {
        "audio/mp3": "http://localhost:8080/nofx-pump_up_the_valuum/04. Dinosaurs Will Die.mp3",
        "audio/ogg": "/track/510/convert?mimetype=audio%2Fogg"
    }
}
```

In that attribute you will find a list of all the supported formats. That list is generated from the `MIMETYPES` setting (see The MIMETYPES config). Just follow the link of the format you need and Shiva will convert it if necessary and serve it for you. As a client, that's all you care about.

But you may have noticed that the URI for the `audio/ogg` format goes through Shiva. This is because the file has not been yet converted, once you call that URI, Shiva will convert the file on the fly, cache it and redirect to the file. The next time the same track is requested, if the file exists it will be served through the file server instead of Shiva:

```
{
    "id": 510,
    "uri": "/track/510",
    "files": {
        "audio/mp3": "http://localhost:8080/nofx-pump_up_the_valuum/04. Dinosaurs Will Die.mp3",
        "audio/ogg": "http://localhost:8080/nofx-pump_up_the_valuum/04. Dinosaurs Will Die.ogg"
    }
}
```

It's completely transparent for the client. If you want an OGG file, you just follow the "audio/ogg" URI blindly, and you will get your file. The first time will take a little longer, though.

8.1 Absolute paths

If you need absolute paths for your `/convert` URIs, just set the `SERVER_URI` setting in your local config, it will be prepended to all the URIs:

```
SERVER_URI = 'http://127.0.0.1:9002'
```

Example output:

```
{
    "files": {
        "audio/mp3": "http://127.0.0.1:8001/flip-keep_rockin/flip-10-away_from_the_sun.mp3",
        "audio/ogg": "http://127.0.0.1:9002/track/1/convert?mimetype=audio%2Fogg"
    },
    "album": {
        "id": 1,
        "uri": "http://127.0.0.1:9002/album/1"
    },
    "length": 168,
    "number": 10,
    "title": "Away From The Sun",
    "slug": "away-from-the-sun",
    "artist": {
        "id": 1,
        "uri": "http://127.0.0.1:9002/artist/1"
    },
    "bitrate": 192000,
    "id": 1,
    "uri": "http://127.0.0.1:9002/track/1"
}
```

Remember to leave out trailing slashes.

8.2 Your converter sucks

So, you don't want to use `ffmpeg`, or you want to call it with different parameters, or chache files differently. That's ok, I won't take it personally.

To overwrite the Converter class to use, just define it in your config:

```
from shiva.myconverter import MyBetterConverter

CONVERTER_CLASS = MyBetterConverter
```

One option is to extend `shiva.converter.Converter` and overwrite the methods that offend you.

The other option is to write a completely new Converter class. If you do so, make sure to have at least the following 3 methods:

- `__init__(Track track, (str, MimeType) mimetype)`: Constructor accepting a path to a file and a mimetype, which could be a string in the form of 'type/subtype', or a MimeType instance.
- `convert()`: Converts to a different format.
- `get_uri()`: Retrieves the URI to the converted file.

The `shiva.resources.ConvertResource` class makes use of them.

8.3 The MimeType class

All mimetypes are represented by a `shiva.media.MimeType` class. The constructor receives the following parameters:

- `type`: Would be `audio` in `audio/ogg`.
- `subtype`: Would be `ogg` in `audio/ogg`.
- `extension`: The extension that converted files should have.
- `acodec` and/or `vcodec`: The codecs used by `Converter.convert()`. Find out the available codecs running:

```
$ ffmpeg -codecs
```

8.4 The MIMETYPES config

You will see in your config file:

```
MIMETYPES = (
    MimeType(type='audio', subtype='mp3', extension='mp3',
             acodec='libmp3lame'),
    MimeType(type='audio', subtype='ogg', extension='ogg',
             acodec='libvorbis'),
)
```

Keep in mind that an invalid MimeType in this config will raise an `InvalidMimeTypeError` exception.

Cross Origin Resource Sharing

CORS support is disabled by default because it's a browser-specific feature, and Shiva doesn't assume that the clients are browsers.

To enable CORS you have to set the following in your `local.py` file:

```
CORS_ENABLED = True
```

Now Shiva will add the following header to each response:

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Accept, Content-Type, Origin, X-Requested-With
```

If you want to limit it to a single origin, then define a tuple with the accepted domains:

```
CORS_ALLOWED_ORIGINS = ('napster.com', 'slsknet.org')
```

Or simply a string:

```
CORS_ALLOWED_ORIGINS = 'napster.com'
```

When a domain (or a tuple of domains) is defined, Shiva will check the request against it. If they match, a header is added:

```
Access-Control-Allow-Origin: http://napster.com
```

CHAPTER 10

Users

Shiva has a very light user implementation. The idea of users is not to keep a complex profile of a person, but to serve as an authentication mechanism.

A user consists of:

- E-mail.
- Password.
- An 'is_public' flag.
- An 'is_active' flag.
- An 'is_admin' flag.
- A creation date.

10.1 User creation

Issue a `POST` request to the `/users` resource. Note the the `GET` method will only list those users whose `is_public` attribute is set to `True`.

```
curl -d "email=herp@derp.com" http://127.0.0.1:9002/users
```

10.2 Authentication

Authentication is done against the `/users/login` endpoint. You will receive a token that, if the `ALLOW_ANONYMOUS_ACCESS` setting is set to `False` (which by default it is), has to be included with every request for as long as it's valid. Once it is no longer valid you will get a `401 Unauthorized` and will have to re-authenticate.

```
curl -d "email=herp@derp.com" -d "password=s3cr37" http://127.0.0.1:9002/users/login
```

It will return something like:

```
{
    "token": "eyJhbGciOiJIUzI1NiIsImV4cCI6MTQxMTUwNDczMywiaWF0IjoxNDExNTAzMjkzfQ.eyJwayI6MX0.7vNzVWGr
}
```

You then need to include that token with your every request:

```
curl http://127.0.0.1:9002/tracks?token=$AUTH_TOKEN
```

CHAPTER 11

Role-based Access Control

The concept of Roles is very limited in Shiva. There are 3 possible roles:

- User
- Admin
- Shiva

The first 2 are assigned to users, the last one is only used by other Shiva instances to communicate with each other. Please note that this functionality is not yet implemented.

To create a normal user (i.e. either *User* or *Admin* roles) use the command *shiva-admin user add*.

A role-authentication failure will result in a 401 Forbidden status code.

CHAPTER 12

Playlists

Playlists are a way of logically grouping tracks by arbitrary conditions defined by the user.

12.1 /playlists

Example response for the request `GET /playlists/1`:

```
{
    "creation_date": "Mon, 29 Sep 2014 22:28:29 -0000",
    "id": 1,
    "length": 4,
    "name": "Classic essentials",
    "read_only": true,
    "tracks": [
        {
            "id": 3,
            "index": 0,
            "uri": "/tracks/3"
        },
        {
            "id": 1,
            "index": 1,
            "uri": "/tracks/1"
        },
        {
            "id": 2,
            "index": 2,
            "uri": "/tracks/2"
        },
        {
            "id": 4,
            "index": 3,
            "uri": "/tracks/4"
        }
    ],
    "user": {
        "id": 1,
        "uri": "/users/1"
    }
}
```

12.1.1 Fields

- `id`: The object's ID.
- `creation_date`: The time the playlist was created.
- `length`: The tracks count.
- `name`: The playlist's name.
- `read_only`: When set to True (the default) only the creator of the playlist can add/remove tracks to it.
- `tracks`: The list of tracks contained by the playlist.
- `tracks.index`: The track's (0-based) position in the playlist.
- `user`: The creator of the playlist.

12.1.2 Filtering

It's possible to get only the playlists for a certain user by using the `user` query parameter: `/playlists?user=1`

12.2 Adding tracks

To add a track to a playlist you have to issue a `POST` request to `/playlists/<id>/add` with the parameters `track`, which is a track id, and `index`, which is the 0-based position the track should occuppy in the playlist. Example:

If the `index` value is largen than the number of items in the playlist, a `400 Bad Request` will be returned. If the playlist is empty, send `index=0`. If all went ok, you will get a `204 No Content` status code.

12.3 Removing tracks

The procedure for removing tracks is very similar to the addition, the main difference is that the request is issued against the `/playlist/<id>/remove` endpoint:

You don't have to include the track id in this case, only the `index` is enough. Just like the addition, if the value of `index` is greater than the number of tracks in the playlist, you will get a `400 Bad Request`. If all went ok, a `204 No Content` will be returned.

CHAPTER 13

Admin utility

The utility `shiva-admin` will let you do some basic management tasks. The following commands are available:

- `user create [<email>]`
- `user activate <email_or_id>`
- `user deactivate <email_or_id>`
- `user delete <email_or_id>`

For more information run `shiva-admin --help`.

CHAPTER 14

Want to contribute?

There are many ways you can contribute:

- File bug reports.
- Implement new features.
- Build your own client.
- Write documentation.
- Write tests.
- Talk about Shiva.
  - Write an article.
  - Give a talk.
- Use it!

If you build a client or write an article about Shiva, let us know and we'll include it in our documentation.

14.1 Sending code

If you want to implement a new feature or fix a bug, remember that every PR that you issue must:

- Strictly follow the PEP8.
- Include documentation, if applicable.
  - Detailed documentation of the new feature.
  - Update old documentation for functionality changes.
- Include tests.
- Not break previous tests.

The CI tool will check for most of this, so make sure the build passes.

14.2 Reporting bugs

Please report bugs, recommend enhancements or ask questions in our issue tracker. Before reporting bugs please make sure of the following:

- The bug was not previously reported.
- You have the latest version installed.
- The bug is not actually a feature.

CHAPTER 15

Wish list

This is a (possible incomplete) list of ideas that may be eventually implemented. With time we will see which of them make sense (or not) and work on them (or not). We may add things that are not documented here as well.

- Index also images and videos.
- Batch-edit ID3 tags.
- Download your tracks in batch.
- Users.
  - Join by invitation
  - Favourite artists.
  - Playlists.
  - Play count.
- Share your music with your friends.
- Share your music with your friends' servers.
- Listen to your friends' music.
- They can also upload their music.
- Stream audio and video. (Radio mode)
- Set up a radio and collaboratively pick the music. (Would this belong to Shiva or to another service consuming Shiva's API?)
- Tabs.

15.1 Assumptions

For the sake of simplicity many assumptions were made that will eventually be worked on and improved/removed.

- Only music files. No videos. No images.
- No update of files' metadata when DB info changes.

Indices and tables

- *genindex*
- *modindex*
- *search*

16.1 What is Shiva?

The Mozilla Hacks blog kindly published a nice article that explains the ideas that inspire this software: Shiva: More than a RESTful API to your music collection.

16.2 Why Shiva?

Shiva is the name of the crater that would have been created by the K-Pg event that extincted the dinosaurs.

16.3 License

Please read the LICENSE file distributed with this software.

16.4 Disclaimer

Remember that when using this software you must comply with your country's laws. You and only you will be held responsible for any law infringement resulting from the misuse of this software.

That said, have fun.