Letterboxd Documentation

Release 0.3.0

Mark Boszko

Mar 17, 2022

Contents:

This software is very much in an alpha state, and things that are working now may break in the future. Developer beware.

CHAPTER 1

Letterboxd

Python 3 implementation of the Letterboxd API v0.

- PyPI package: https://pypi.org/project/letterboxd/
- GitHub repo: https://github.com/bobtiki/letterboxd
- Documentation: https://letterboxd.readthedocs.io
- Free software: MIT license

Python 3.6 is required.

Warning: THIS PROJECT IS CURRENTLY IN ALPHA:

- It may be broken.
- What is working now may break between now and v1.0
- Initial focus is on implementing endpoints related to retrieving watchlists and other lists for users.

1.1 Letterboxd API Access

Letterboxd has posted an example Ruby client, but as they say in the readme there:

> Although the Letterboxd API isn't public yet (as at 2017-06-12), we have seeded some developers with API keys.

If you need more information about API access, please see https://letterboxd.com/api-coming-soon/.

CHAPTER 2

Installation

2.1 Stable release

To install Letterboxd, run this command in your terminal:

```
$ pip install letterboxd
```

This is the preferred method to install Letterboxd, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Letterboxd can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/bobtiki/letterboxd
```

Or download the tarball:

```
$ curl  -OL https://github.com/bobtiki/letterboxd/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use Letterboxd in a project:

```
import letterboxd
lbxd = letterboxd.new()
```

You can pass the `api_base` URL for the API, your `api_key`, and your `api_secret` as arguments in `.new()`, or add it to your environment variables as:

```
export LBXD_API_KEY="YOUR_KEY_HERE"
export LBXD_API_SECRET="YOUR_SECRET_HERE"
```

If you do not provide an `api_base`, it currently defaults to `https://api.letterboxd.com/api/v0`.

For testing, you can also add environment variables for an account user name and password, and the location of your Charles proxy SSL certificate.

```
export LBXD_USERNAME="YOUR_USERNAME"
export LBXD_PASSWORD="YOUR_PASSWORD"
export CHARLES_CERTIFICATE="/path/to/charles-ssl-proxying-certificate.pem"
```

Set the `CHARLES` environment variable to `True` to turn the proxy settings on. For example, if you want to use it on a case-by-case basis for testing, run the tests like this:

```
$ CHARLES="True" pytest
```

Please see Modules for more details on usage of the classes.

CHAPTER 4

Modules reference

CHAPTER 5

letterboxd package reference

5.1 letterboxd module

Python 3 wrapper for Version 0 of the Letterboxd API

class letterboxd.letterboxd.**Letterboxd**(*api_base='https://api.letterboxd.com/api/v0'*, *api_key=''*, *api_secret=''*)

Loads the API base URL, API key, and API shared secret, and connects with all of the other classes.

If the key and secret are not passed as arguments, it looks for them as environment variables, as LBXD_API_KEY and LBXD_API_SECRET.

auth()

Returns services.auth.Authentication object

film(*film_id*)

Parameters **film_id** – str - the LID of a film on Letterboxd

Returns services.film.Film object

film_collection(*film_collection_id*, *film_collection_request*)
/film-collection/{id}

Get details about a film collection by ID. The response will include the film relationships for the signed-in member and the member indicated by the member LID if specified.

Parameters

- **film_collection_id** – str - LID of the FilmCollection
- **film_collection_request** – dict - FilmCollectionRequest

Returns dict - FilmCollection

films()

Returns services.film.Films object

list(*list_id*)

Parameters **list_id** – str - the LID of a list on Letterboxd

Returns services.list.List object

member(*member_id*)

Parameters **member_id** – str - LID for Letterboxd member

Returns services.member.Member object

search(*search_request*)
/search

Parameters **search_request** – dict - SearchRequest

Returns dict - SearchResponse

user(*username*, *password*)
Signs in the user, and adds the oAuth token to future API calls

Parameters

- **username** – str
- **password** – str

Returns user.User object

5.2 api module

class letterboxd.api.**API**(*api_base*, *api_key*, *api_secret*)
Communication methods for the Letterboxd API

api_call(*path*, *params={}*, *form=None*, *headers={}*, *method='get'*)
The workhorse method of calls to the Letterboxd API

Parameters

- **path** – str - URL endpoint path for the desired service
- **params** – dict - request parameters
- **form** – str - form information, likely from the auth.py call
- **headers** – dict - request parameters
- **method** – str - HTML methods, [get, post, put, patch, delete]

Returns requests.Response object

5.3 user module

User-based features of the Letterboxd API

class letterboxd.user.**User**(*api*, *username*, *password*)
Provices access token and shortcuts to user-focused methods

me
/me

Get details about the authenticated member.

Calls to this endpoint must include the access token for an authenticated member.

Returns dict - MemberAccount

me_update(*member_settings_update_request*)
/me

Update the profile settings for the authenticated member.

Calls to this endpoint must include the access token for an authenticated member

Parameters **member_settings_update_request** – dict - MemberSettingsUpdateRequest

Returns dict - MemberSettingsUpdateResponse

token
Ask services.auth to get a token, and return the token string

Returns str - oAuth token

5.4 config module

Configuration

Constants for package use.

CHAPTER 6

services package reference

6.1 auth module

User authentication services for the Letterboxd API

Authentication API Documentation: http://api-docs.letterbotokend.com/#auth

class letterboxd.services.auth.**Authentication**(*api*, *username*, *password*)
User authentication services for Letterboxd

This token business mostly takes care of itself. Instantiate authentication with username and password, then call token(), and if there isn't a token already, or if it's expired, it will go and get one.

static forgotten_password_request(*api*, *forgotten_password_request*)
/auth/forgotten-password-request

Request a link via email to reset the password for a member's account.

Request forgotten_password_request - ForgottenPasswordRequest

Returns int - HTTP status code

login(*username*, *password*)
User access to the Letterboxd API. Requests a token for the user.

Parameters

- **username** – str
- **password** – str

Returns dict - either an AccessToken or OAuthError

refresh_token()
Uses the current single-use refresh_token to request a new access token for the user

Returns dict - either an AccessToken or OAuthError

token
Checks if the user authentication token already exists. If not, it tries to get one. If it does exist, it checks to see if it is expired, and if so, it attempts to refresh the token.

Returns str - user token

6.2 film module

class letterboxd.services.film.**Film**(*api*, *film_id=None*)
/film/* services for the Letterboxd API

availability(*film_id=None*)
/film/{id}/availability

Get availability data about a film by ID. If no film ID passed, uses the initialized film.

NOTE: This data is currently available to first-party only.

Parameters **film_id** – str - LID of the film

Returns dict - FilmAvailabilityResponse

details(*film_id=None*)
/film/{id}

Get details about a film by ID. If no film ID passed, uses the initialized film.

Parameters **film_id** – str - LID of the film

Returns dict - Film

me(*film_id=None*)
/film/{id}/me

Get details of the authenticated member's relationship with a film by ID. If no film ID passed, uses the initialized film.

Parameters **film_id** – str - LID of the film

Returns dict - FilmRelationship

me_update(*film_relationship_update_request*, *film_id=None*)
/film/{id}/me [PATCH]

Update the authenticated member's relationship with a film by ID.

Calls to this endpoint must include the access token for an authenticated member

Parameters

- **film_id** – str - LID of the film
- **film_relationship_update_request** – dict - FilmRelationshipUpdateRequest

Returns dict - FilmRelationshipUpdateResponse

members(*film_id=None*, *member_film_relationships_request=None*)
/film/{id}/members

Get details of members' relationships with a film by ID. If no film ID passed, uses the initialized film.

Parameters

- **film_id** – str - LID of the film

- **member_film_relationships_request** – dict - MemberFilmRelationshipsRequest

Returns dict - MemberFilmRelationshipsResponse

report(*film_id=None*, *report_film_request=None*)
/film/{id}/report

Report problems with a film by ID. Does NOT default to the initialized Film instance LID, so as to not submit unnecessary reports.

Parameters

- **film_id** – str - the LID of the film
- **report_film_request** – dict - ReportFilmRequest

Returns requests.Response.status_code

statistics(*film_id=None*)
/film/{id}/statistics

Get statistical data about a film by ID.

Parameters **film_id** – str - the LID of the film

Returns dict - FilmStatistics

class letterboxd.services.film.**FilmCollection**(*api*)
/film-collection service for the Letterboxd API

film_collection(*film_collection_id=None*, *film_collection_request=None*)
/film-collection/{id}

Get details about a film collection by ID. The response will include the film relationships for the signed-in member and the member indicated by the member LID if specified.

Parameters

- **film_collection_id** – str - The LID of the film collection.
- **film_collection_request** – dict - FilmCollectionRequest

Returns dict - FilmCollection

class letterboxd.services.film.**Films**(*api*)
/films/* services for the Letterboxd API

films(*films_request=None*)
/films

A cursored window over the list of films.

Use the 'next' cursor to move through the list. The response will include the film relationships for the signed-in member and the member indicated by the member LID if specified.

Parameters **films_request** – dict - FilmsRequest

Returns dict

genres()
/films/genres

Get a list of genres supported by the /films endpoint.

Genres are returned in alphabetical order.

Returns dict - GenresResponse

services()
/films/film-services

Get a list of services supported by the /films endpoint.

Services are returned in alphabetical order. Some services are only available to paying members, so results will vary based on the authenticated member's status.

Returns dict - FilmServicesResponse

6.3 member module

class letterboxd.services.member.**Member**(*api*, *member_id=None*)
/member/* services for the Letterboxd API

details(*member_id=None*)
/member/{id}

Get details about a member by ID.

TODO: Write this function

Parameters **member_id** – str - The LID of the member.

Returns dict - Member

watchlist(*member_id=None*, *watchlist_request=None*)
/member/{id}/watchlist

Get details of a member's public watchlist by ID.

The response will include the film relationships for the signed-in member, the watchlist's owner, and the member indicated by the member LID if specified (the member and memberRelationship parameters are optional, and can be used to perform comparisons between the watchlist owner and another member). Use the /film/{id}/me endpoint to add or remove films from a member's watchlist.

Parameters

- **member_id** – str - The LID of the member.
- **watchlist_request** – dict - WatchlistRequest

Returns dict - FilmsResponse

CHAPTER 7

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:

7.1 Types of Contributions

7.1.1 Report Bugs

Report bugs at https://github.com/bobtiki/letterboxd/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.

7.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

7.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

7.1.4 Write Documentation

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

7.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/bobtiki/letterboxd/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 :)

7.2 Get Started!

Ready to contribute? Here's how to set up *letterboxd* for local development.

1. Fork the *letterboxd* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/letterboxd.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 letterboxd
$ cd letterboxd/
$ 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, run it through the Black code reformatter, and check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ black letterboxd
$ flake8 letterboxd tests
$ python setup.py test or pytest
$ 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.

7.3 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 3.6 and 3.7. (Once PyPy adds support for Python 3.6, the pull request should also work for `pypy3`.) Check https://travis-ci.org/bobtiki/letterboxd/pull_requests and make sure that the tests pass for all supported Python versions.
4. Use the pull request template to format the description and changelog information for your pull request.

7.4 Tips

To run a subset of tests:

```
$ pytest tests.test_letterboxd
```

7.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in CHANGELOG.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 8

Credits

8.1 Development Lead

- Mark Boszko <mboszko@mac.com>

8.2 Contributors

- Chris Pruitt, advice and feedback

CHAPTER 9

Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning.

9.1 [Unreleased]

9.2 [0.3.0] - 2018-07-22

9.2.1 ADDED

- Issue and pull request templates
- Code of Conduct
- **Coverage for Letterboxd API endpoints**
  - /auth/forgotten-password-request
  - /auth/username-check
  - /list/{id}
  - /list/{id} [PATCH]
  - /list/{id} [DELETE]
  - /list/{id}/comments
  - /list/{id}/comments [POST]
  - /list/{id}/entries
  - /list/{id}/me
  - /list/{id}/me [PATCH]

  - /list/{id}/report [POST]
  - /list/{id}/statistics
  - /lists
  - /lists [POST]
  - /me [PATCH]
- A TON more pytest unit tests, and made existing ones more comprehensive

9.2.2 CHANGED

- Internally refactored API object definitions into their own file, for pytest
- Internally changed some variable names to better reflect Letterboxd API nomenclature (I don't *think* this affects any method arguments.)

9.2.3 FIXED

- Removed mutable default arguments on several methods

9.3 [0.2.6] - 2018-07-04

9.3.1 CHANGED

- Getting the Travis CI integration with PyPI to work properly.

9.4 [0.2.5] - 2018-07-04 [YANKED]

9.5 [0.2.4] - 2018-07-04

9.5.1 CHANGED

- Getting `bumpversion` to work properly.

9.6 [0.2.3] - 2018-07-04 [YANKED]

9.7 [0.2.2] - 2018-07-04 [YANKED]

9.8 [0.2.1] - 2018-07-04 [YANKED]

9.9 [0.2.0] - 2018-07-04

9.9.1 Added

- This `CHANGELOG.rst`
- Converted `README.md` to `.rst`
- Documentation written with reStructuredText and Sphinx, being built to Read the Docs
- Added a number of defaults and tests as provided in cookiecutter-pypackage
- Added an easy initializer with `import letterboxd` and then `letterboxd.new()`
- `User.refresh_token()` to refresh the user authentication oAuth token
- **Coverage for Letterboxd API endpoints:**
  - /film/{id}/members
  - /film/{id}/report
  - /film/{id}/statistics
  - /films
  - /films/film-services
  - /films/genres
  - /film-collection/{id}
  - /search

9.9.2 Changed

- All api-calling methods now return the dictionary from the response JSON, instead of the entire `requests.Response`.

9.10 [0.1.0] - 2018-06-24

9.10.1 Added

- First public version! Version 0.1.0a tagged on GitHub, and posted to PyPI.
- letterboxd, api, user, auth, member, and film modules.
- **coverage for Letterboxd API endpoints:**
  - film

  - /film/{id}
  - /film/{id}/availability — this data is first-party only
  - /film/{id}/me
  - /me
  - /member/{id}/watchlist

9.11 Changelog format

- **Each version should:**
  - List its release date in ISO 8601 format (YYYY-MM-DD).
  - **Group changes to describe their impact on the project, as follows:**
    - `Added` for new features.
    - `Changed` for changes in existing functionality.
    - `Deprecated` for once-stable features removed in upcoming releases.
    - `Removed` for deprecated features removed in this release.
    - `Fixed` for any bug fixes.
    - `Security` to invite users to upgrade in case of vulnerabilities.
  - Take a look at this checklist for packaging a new version, and this one

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index

W