stravapi Documentation

Release 0.0.2

Oriol Fabregas

Mar 11, 2019

Contents

Contents:

CHAPTER 1

stravapi

A portable RestAPI for Strava.

- Documentation: https://stravapi.readthedocs.org/en/latest

1.1 Project Features

This is a RestAPI for Strava made with Hug. This is for learning purposes and it also adds some useful endpoints not supported by native Strava API.

Check out the installation instructions.

CHAPTER 2

Installation

Make sure you have xcode installed (MacOS only)

First off, it's recommended to create a virtualenv. If you don't know how, you, can follow these instructions:

Once your virtualenv is activated, install the required packages:

```
$ pip install stravapi
$ pip install uwsgi
```

If you run into `openssl` problems on MacOS, you may need to run the following:

```
$ brew install openssl
```

Now you are going to need to configure `pystrava` which it will handle the authentication to Strava API. You can configure it by following this link.

Basically you will need to export the following environment variables as this API does not support other methods at the moment:

```
$ export CLIENT_ID=
$ export SECRET=
$ export CALLBACK_URL=
$ export SCOPE=
$ export EMAIL=
$ export PASSWORD=
```

Clone this repository as follows:

```
$ git clone https://github.com/wefner/stravapi.git
```

and then you should be able to run:

```
$ cd stravapi/
$ hug -f stravapi/stravapi.py
```

Or if you prefer to run it with `uwsgi`:

```
$ uwsgi --http 0.0.0.0:8000 \
    --wsgi-file $HOME/.pyenv/versions/<your_version>/envs/uwsgi/lib/python<your_
↪version>/site-packages/stravapi/stravapi.py \
    --callable \
    __hug_wsgi__
```

Both commands will expose a RestAPI on port `8000`. You can open a browser and try different activities with the given URLs:

```
- http://localhost:8000/activities?activity_type=swim&number=4
- http://localhost:8000/activities?activity_type=ride&number=1
- http://localhost:8000/activities?activity_type=run&number=10
```

CHAPTER 3

Usage

To develop on stravapi:

```
# The following commands require pipenv as a dependency

# To lint the project
_CI/scripts/lint.py

# To execute the testing
_CI/scripts/test.py

# To create a graph of the package and dependency tree
_CI/scripts/graph.py

# To build a package of the project under the directory "dist/"
_CI/scripts/build.py

# To see the package version
_CI/scipts/tag.py

# To bump semantic versioning [--major|--minor|--patch]
_CI/scipts/tag.py --major|--minor|--patch

# To upload the project to a pypi repo if user and password are properly provided
_CI/scripts/upload.py

# To build the documentation of the project
_CI/scripts/document.py
```

To use stravapi in a project:

```
from stravapi import Strava_api
stravapi = Strava_api()
```

CHAPTER 4

Contributing

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

4.1 Submit Feedback

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.

4.1.1 Development Workflow

The workflow supports the following steps

- lint
- test
- build
- document
- upload
- graph

These actions are supported out of the box by the corresponding scripts under _CI/scripts directory with sane defaults based on best practices. Sourcing setup_aliases.ps1 for windows powershell or setup_aliases.sh in bash on Mac or Linux will provide with handy aliases for the shell of all those commands prepended with an underscore.

The bootstrap script creates a .venv directory inside the project directory hosting the virtual environment. It uses pipenv for that. It is called by all other scripts before they do anything. So one could simple start by calling _lint and that would set up everything before it tried to actually lint the project

Once the code is ready to be delivered the _tag script should be called accepting one of three arguments, patch, minor, major following the semantic versioning scheme. So for the initial delivery one would call

> $ _tag –minor

which would bump the version of the project to 0.1.0 tag it in git and do a push and also ask for the change and automagically update HISTORY.rst with the version and the change provided.

So the full workflow after git is initialized is:

- repeat as necessary (of course it could be test - code - lint :)) * code * lint * test
- commit and push
- develop more through the code-lint-test cycle
- tag (with the appropriate argument)
- build
- upload (if you want to host your package in pypi)
- document (of course this could be run at any point)

4.1.2 Important Information

This template is based on pipenv. In order to be compatible with requirements.txt so the actual created package can be used by any part of the existing python ecosystem some hacks were needed. So when building a package out of this **do not** simple call

> $ python setup.py sdist bdist_egg

as this will produce an unusable artifact with files missing. Instead use the provided build and upload scripts that create all the necessary files in the artifact.

Get Started!

Ready to contribute? Here's how to set up *stravapi* for local development. Using of pipenv is highly recommended.

1. Clone your fork locally:

```
$ git clone https://github.com/wefner/stravapi
```

2. Install your local copy into a virtualenv. Assuming you have pipenv installed, this is how you set up your clone for local development:

```
$ cd stravapi/
$ pipenv install --ignore-pipfile
```

3. Create a branch for local development:

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

   Now you can make your changes locally. Do your development while using the CI capabilities and making sure the code passes lint, test, build and document stages.

4. Commit your changes and push your branch to the server:

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

5. Submit a merge request

CHAPTER 5

stravapi

5.1 stravapi package

5.1.1 Submodules

5.1.2 stravapi.client module

class `stravapi.client.`**`Client`**(*limit=500*)

Bases: `object`

Strava object to wrap authentication and activities.

`activities`

Gets all activities for a given user.

Limit is set when initializing and cannot be overwritten here. :return: list of Activity objects

`strava`

Client authenticated.

Caching the object so we don't call it every time. :return:

5.1.3 stravapi.stravapi module

Main code for stravapi

`stravapi.stravapi.`**`activities`**

Defines the Interface responsible for exposing functions locally

5.1.4 stravapi.stravapiexceptions module

Custom exception code for stravapi

5.1.5 Module contents

stravapi package

Import all parts from stravapi here

CHAPTER 6

Credits

6.1 Development Lead

- Oriol Fabregas <fabregas.oriol@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

CHAPTER 8

0.0.2 (11-03-2019)

- Updated docs

CHAPTER 9

0.0.1 (01-03-2019)

- First code creation

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index