# Flask-EasyAPI Documentation

***Release 0.1.1***

**Harsh Parekh**

**Feb 14, 2022**

# Contents:

CHAPTER 1

# Flask-EasyAPI

Rest API on Flask made a little too easy.

- Documentation: https://flask-easyapi.readthedocs.io.

## 1.1 Features

- One to one mapping from functions to api endpoints via decorators

### 1.1.1 Planned

- Use type annotations to automatically add type-checks to parameters recieved in the requests to the api.
- Return HTTP error codes and error messages by raising exceptions.

# CHAPTER 2

# Installation

## 2.1 Stable release

To install Flask-EasyAPI, run this command in your terminal:

```
$ pip install flask_easyapi
```

This is the preferred method to install Flask-EasyAPI, 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 Flask-EasyAPI can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/hXtreme/flask_easyapi
```

Or download the tarball:

```
$ curl -OJL https://github.com/hXtreme/flask_easyapi/tarball/master
```

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

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

# CHAPTER 3

# Usage

To use Flask-EasyAPI in a project:

```
import flask_easyapi
```

CHAPTER 4

# flask_easyapi

## 4.1 flask_easyapi package

### 4.1.1 Submodules

### 4.1.2 flask_easyapi.flask_easyapi module

**class** `flask_easyapi.flask_easyapi.`**`EasyAPI`**(*name: str*, *import_name: str*, **args*, ***kwargs*)

Bases: `flask.blueprints.Blueprint`

EasyAPI is an object that makes defining a collection of related rest-api easier.

Represents a collection of related rest-api routes that can later be registered on a real application.

**Parameters**

- **`name`** – The name of the blueprint. Will be prepended to each endpoint name.
- **`import_name`** – The name of the blueprint package, usually `__name__`. This helps locate the `root_path` for the blueprint.

**Keyword arguments passed to Blueprint**

See flask Blueprint api for up-to-date information. The following is an extract of docs under BSD-3-Clause License:

**Parameters**

- **`static_folder`** – A folder with static files that should be served by the blueprint's static route. The path is relative to the blueprint's root path. Blueprint static files are disabled by default.
- **`static_url_path`** – The url to serve static files from. Defaults to `static_folder`. If the blueprint does not have a `url_prefix`, the app's static route will take precedence, and the blueprint's static files won't be accessible.

- **template_folder** – A folder with templates that should be added to the app's template search path. The path is relative to the blueprint's root path. Blueprint templates are disabled by default. Blueprint templates have a lower precedence than those in the app's templates folder.
- **url_prefix** – A path to prepend to all of the blueprint's URLs, to make them distinct from the rest of the app's routes.
- **subdomain** – A subdomain that blueprint routes will match on by default.
- **url_defaults** – A dict of default values that blueprint routes will receive by default.
- **root_path** – By default, the blueprint will automatically this based on `import_name`. In certain situations this automatic detection can fail, so the path can be specified manually instead.

**route**(*rule: str*, ***options*)

A decorator that is used to register an api endpoint and its handler. The decorated function will automatically receive the url parameters as kwargs.

---

**Note:** As of v0.1.0 unlike Blueprint, `route()` and `add_url_rule()` behaves differently for EasyAPI, this difference is expected to disappear in later releases.

---

**Parameters** **rule** – The URL rule as string. See flask route registrations api

**Keyword arguments passed to Blueprint**

See flask route api for up-to-date information. The following is an extract of docs under BSD-3-Clause License:

**Parameters**

- **endpoint** – the endpoint for the registered URL rule. Flask itself assumes the name of the view function as endpoint
- **options** – the options to be forwarded to the underlying `Rule` object. A change to Werkzeug is handling of method options. methods is a list of methods this rule should be limited to (`GET`, `POST` etc.). By default a rule just listens for `GET` (and implicitly `HEAD`). Starting with Flask 0.6, `OPTIONS` is implicitly added and handled by the standard request handling.

## 4.1.3 Module contents

### Flask-EasyAPI

EasyAPI is a flask extension that aims to make writing rest-api with flask very easy.

COPYRIGHT: Harsh Parekh LICENSE: MIT

CHAPTER 5

# 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:

## 5.1 Types of Contributions

### 5.1.1 Report Bugs

Report bugs at https://github.com/hXtreme/flask_easyapi/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.
- Expected and Actual behaviour of your code.

### 5.1.2 Fix Bugs

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

### 5.1.3 Implement Features

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

#### Other Issues

Look through Github issues for open issues. Things tagged "help wanted" are open to anyone; for others confirm with project team to pick up the issue.

### 5.1.4 Write Documentation

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

### 5.1.5 Submit Feedback

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

## 5.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/flask_easyapi.git
```

3. Install your local copy into a virtualenv. This is how you set up your fork for local development:

```
$ cd flask_easyapi/
$ virtualenv env && . ./env/bin/activate
$ pip installl -r requirements_dev.txt
$ pip install --editable .
```

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, make sure to follows black's style guide and check that your changes pass flake8, the tests, including testing other Python versions with tox:

```
$ black flask_easyapi tests
$ flake8 flask_easyapi tests
$ pytest
$ tox
```

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.

## 5.3 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, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/hXtreme/flask_easyapi/pull_requests and make sure that the tests pass for all supported Python versions.

## 5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_flask_easyapi
```

## 5.5 Deploying

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

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Github workflow will then deploy to PyPI if tests pass.

# CHAPTER 6

# Credits

## 6.1 Development Lead

- hXtreme <harsh_parekh@outlook.com> : Project manager.

## 6.2 Contributors

None yet. Why not be the first?

## 6.3 Special Thanks

- audreyr : For the cookiecutter template.

# CHAPTER 7

# History

## 7.1 0.1.1 (2020-08-28)

- Update README to reflect implemented feature.

## 7.2 0.1.0 (2020-08-28)

- Add EasyAPI class that functions as a Blueprint for a set of api call routes.
  - Implement EasyAPI.route(.) decorator.
- Add a example/test Flask hello world greeting app.
  - Add tests for EasyAPI using the greet app.
- Add and update documentation.
- Refactoring and linting changes.
- Fix typos.

## 7.3 0.0.6 (2020-08-21)

- Fix project url in setup.py

## 7.4 0.0.5 (2020-08-20)

- Improve CD process.

## 7.5 0.0.4 (2020-08-19)

- Improve CD process.

## 7.6 0.0.3 (2020-08-19)

- Various changes to CI configuration.
- Update dependencies.
- Edit README.rst to add badges and alt-text.

## 7.7 0.0.2 (2020-08-19)

- Fix badges in readme
- Add pyup configuration

## 7.8 0.0.1 (2020-08-19)

- Fix minor issues and typos to make tests pass.

## 7.9 0.0.0 (2020-08-19)

- First release on PyPI. [Falied]

# CHAPTER 8

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

# Index