Flask-Cors Documentation

Release 2.1.0

Cory Dolphin

August 07, 2015

Contents

A Flask extension for handling Cross Origin Resource Sharing (CORS), making cross-origin AJAX possible.

CHAPTER 1

Installation

Install the extension with using pip, or easy_install.

```
$ pip install -U flask-cors
```

CHAPTER 2

Usage

This extension enables CORS support either via a decorator, or a Flask extension. There are three examples shown in the examples directory, showing the major use cases. The suggested configuration is the simple_example.py, or the app_example.py.

2.1 Simple Usage

In the simplest case, initialize the Flask-Cors extension with default arguments in order to allow CORS on all routes.

```
app = Flask(__name__)
cors = CORS(app)

@app.route("/")
def helloWorld():
  return "Hello, cross-origin-world!"
```

2.1.1 Resource specific CORS

Alternatively, a list of resources and associated settings for CORS can be supplied, selectively enables CORS support on a set of paths on your app.

Note: this resources parameter can also be set in your application's config.

```
app = Flask(__name__)
cors = CORS(app, resources={r"/api/*": {"origins": "*"}})

@app.route("/api/v1/users")
def list_users():
  return "user example"
```

2.1.2 Route specific CORS via decorator

This extension also exposes a simple decorator to decorate flask routes with. Simply add `@cross_origin()` below a call to Flask's `@app.route(..)` incanation to accept the default options and allow CORS on a given route.

```
@app.route("/")
@cross_origin() # allow all origins all methods.
def helloWorld():
  return "Hello, cross-origin-world!"
```

2.1.3 Logging

Flask-Cors uses standard Python logging, using the logger name 'app.logger_name.cors'. The app's logger name attribute is usually the same as the name of the app. You can read more about logging from Flask's documentation.

```
import logging
# make your awesome app
logging.basicConfig(level=logging.INFO)
```

2.2 CORS the Extension

class flask_cors.**CORS**(*app=None*, ***kwargs*)

Initializes Cross Origin Resource sharing for the application. The arguments are identical to cross_origin(), with the addition of a *resources* parameter. The resources parameter defines a series of regular expressions for resource paths to match and optionally, the associated options to be applied to the particular resource. These options are identical to the arguments to cross_origin().

The settings for CORS are determined in the following order: Resource level settings (e.g when passed as a dictionary) Keyword argument settings App level configuration settings (e.g. CORS_*) Default settings

Note: as it is possible for multiple regular expressions to match a resource path, the regular expressions are first sorted by length, from longest to shortest, in order to attempt to match the most specific regular expression. This allows the definition of a number of specific resource options, with a wildcard fallback for all other resources.

Parameters **resources** – the series of regular expression and (optionally)

associated CORS options to be applied to the given resource path.

If the argument is a dictionary, it is expected to be of the form: regexp : dict_of_options

If the argument is a list, it is expected to be a list of regular expressions, for which the app-wide configured options are applied.

If the argument is a string, it is expected to be a regular expression for which the app-wide configured options are applied.

Default :'*'

2.2.1 A simple example

This is the suggested approach to enabling CORS. The default configuration will work well for most use cases.

```
"""
Flask-Cors example
==================
This is a tiny Flask Application demonstrating Flask-Cors, making it simple
to add cross origin support to your flask app!

:copyright: (C) 2013 by Cory Dolphin.
:license:   MIT/X11, see LICENSE for more details.
"""
from flask import Flask, jsonify
import logging
try:
    from flask.ext.cors import CORS  # The typical way to import flask-cors
```

```
except ImportError:
    # Path hack allows examples to be run without installation.
    import os
    parentdir = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
    os.sys.path.insert(0, parentdir)

    from flask.ext.cors import CORS


app = Flask('FlaskCorsAppBasedExample')
logging.basicConfig(level=logging.INFO)

# One of the simplest configurations. Exposes all resources matching /api/* to
# CORS and allows the Content-Type header, which is necessary to POST JSON
# cross origin.
CORS(app, resources=r'/api/*', allow_headers='Content-Type')


@app.route("/")
def helloWorld():
    '''
        Since the path '/' does not match the regular expression r'/api/*',
        this route does not have CORS headers set.
    '''
    return '''<h1>Hello CORS!</h1> Read about my spec at the
<a href="http://www.w3.org/TR/cors/">W3</a> Or, checkout my documentation
on <a href="https://github.com/corydolphin/flask-cors">Github</a>'''


@app.route("/api/v1/users/")
def list_users():
    '''
        Since the path matches the regular expression r'/api/*', this resource
        automatically has CORS headers set. The expected result is as follows:

        $ curl --include -X GET http://127.0.0.1:5000/api/v1/users/ \
            --header Origin:www.examplesite.com
        HTTP/1.0 200 OK
        Access-Control-Allow-Headers: Content-Type
        Access-Control-Allow-Origin: *
        Content-Length: 21
        Content-Type: application/json
        Date: Sat, 09 Aug 2014 00:26:41 GMT
        Server: Werkzeug/0.9.4 Python/2.7.8

        {
            "success": true
        }

    '''
    return jsonify(user="joe")


@app.route("/api/v1/users/create", methods=['POST'])
def create_user():
    '''
        Since the path matches the regular expression r'/api/*', this resource
        automatically has CORS headers set.
```

```
        Browsers will first make a preflight request to verify that the resource
        allows cross-origin POSTs with a JSON Content-Type, which can be simulated
        as:
        $ curl --include -X OPTIONS http://127.0.0.1:5000/api/v1/users/create \
            --header Access-Control-Request-Method:POST \
            --header Access-Control-Request-Headers:Content-Type \
            --header Origin:www.examplesite.com
        >> HTTP/1.0 200 OK
        Content-Type: text/html; charset=utf-8
        Allow: POST, OPTIONS
        Access-Control-Allow-Origin: *
        Access-Control-Allow-Headers: Content-Type
        Access-Control-Allow-Methods: DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT
        Content-Length: 0
        Server: Werkzeug/0.9.6 Python/2.7.9
        Date: Sat, 31 Jan 2015 22:25:22 GMT


        $ curl --include -X POST http://127.0.0.1:5000/api/v1/users/create \
            --header Content-Type:application/json \
            --header Origin:www.examplesite.com


        >> HTTP/1.0 200 OK
        Content-Type: application/json
        Content-Length: 21
        Access-Control-Allow-Origin: *
        Server: Werkzeug/0.9.6 Python/2.7.9
        Date: Sat, 31 Jan 2015 22:25:04 GMT

        {
          "success": true
        }

    '''
    return jsonify(success=True)


if __name__ == "__main__":
    app.run(debug=True)
```

2.3 CORS the Decorator

flask_cors.**cross_origin**(**args*, ***kwargs*)

This function is the decorator which is used to wrap a Flask route with. In the simplest case, simply use the default parameters to allow all origins in what is the most permissive configuration. If this method modifies state or performs authentication which may be brute-forced, you should add some degree of protection, such as Cross Site Forgery Request protection.

Parameters

- **origins** (*list, string or regex*) – The origin, or list of origins to allow requests from. The origin(s) may be regular expressions, case-sensitive strings, or else an asterisk

  Default : '*'

- **methods** (*list or string*) – The method or list of methods which the allowed origins are allowed to access for non-simple requests.

  Default : [GET, HEAD, POST, OPTIONS, PUT, PATCH, DELETE]

- **expose_headers** (*list or string*) – The header or list which are safe to expose to the API of a CORS API specification.

  Default : None

- **allow_headers** (*list, string or regex*) – The header or list of header field names which can be used when this resource is accessed by allowed origins. The header(s) may be regular expressions, case-sensitive strings, or else an asterisk.

  Default : '*', allow all headers

- **supports_credentials** (*bool*) – Allows users to make authenticated requests. If true, injects the *Access-Control-Allow-Credentials* header in responses. This allows cookies to be submitted across domains.

  note This option cannot be used in conjuction with a '*' origin

  Default : False

- **max_age** (*timedelta, integer, string or None*) – The maximum time for which this CORS request maybe cached. This value is set as the *Access-Control-Max-Age* header.

  Default : None

- **send_wildcard** (*bool*) – If True, and the origins parameter is *, a wildcard *Access-Control-Allow-Origin* header is sent, rather than the request's *Origin* header.

  Default : False

- **vary_header** (*bool*) – If True, the header Vary: Origin will be returned as per the W3 implementation guidelines.

  Setting this header when the *Access-Control-Allow-Origin* is dynamically generated (e.g. when there is more than one allowed origin, and an Origin than '*' is returned) informs CDNs and other caches that the CORS headers are dynamic, and cannot be cached.

  If False, the Vary header will never be injected or altered.

  Default : True

- **automatic_options** (*bool*) – Only applies to the *cross_origin* decorator. If True, Flask-CORS will override Flask's default OPTIONS handling to return CORS headers for OPTIONS requests.

  Default : True

2.3.1 A view-based example

Alternatively, using the decorator on a per view basis enables CORS for only a particular view.

```
"""
Flask-Cors example
==================
This is a tiny Flask Application demonstrating Flask-Cors, making it simple
to add cross origin support to your flask app!

:copyright: (C) 2013 by Cory Dolphin.
:license:   MIT/X11, see LICENSE for more details.
```

```
"""
from flask import Flask, jsonify
import logging
try:
    # The typical way to import flask-cors
    from flask.ext.cors import cross_origin
except ImportError:
    # Path hack allows examples to be run without installation.
    import os
    parentdir = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
    os.sys.path.insert(0, parentdir)

    from flask.ext.cors import cross_origin


app = Flask('FlaskCorsViewBasedExample')
logging.basicConfig(level=logging.INFO)

@app.route("/", methods=['GET'])
@cross_origin()
def helloWorld():
    '''
        This view has CORS enabled for all domains, representing the simplest
        configuration of view-based decoration. The expected result is as
        follows:

        $ curl --include -X GET http://127.0.0.1:5000/ \
            --header Origin:www.examplesite.com

        >> HTTP/1.0 200 OK
        Content-Type: text/html; charset=utf-8
        Content-Length: 184
        Access-Control-Allow-Origin: *
        Server: Werkzeug/0.9.6 Python/2.7.9
        Date: Sat, 31 Jan 2015 22:29:56 GMT

        <h1>Hello CORS!</h1> Read about my spec at the
        <a href="http://www.w3.org/TR/cors/">W3</a> Or, checkout my documentation
        on <a href="https://github.com/corydolphin/flask-cors">Github</a>

    '''
    return '''<h1>Hello CORS!</h1> Read about my spec at the
<a href="http://www.w3.org/TR/cors/">W3</a> Or, checkout my documentation
on <a href="https://github.com/corydolphin/flask-cors">Github</a>'''


@app.route("/api/v1/user/create", methods=['GET', 'POST'])
@cross_origin(allow_headers=['Content-Type'])
def cross_origin_json_post():
    '''
        This view has CORS enabled for all domains, and allows browsers
        to send the Content-Type header, allowing cross domain AJAX POST
        requests.

 Browsers will first make a preflight request to verify that the resource
        allows cross-origin POSTs with a JSON Content-Type, which can be simulated
        as:
        $ curl --include -X OPTIONS http://127.0.0.1:5000/api/v1/users/create \
```

```
            --header Access-Control-Request-Method:POST \
            --header Access-Control-Request-Headers:Content-Type \
            --header Origin:www.examplesite.com
        >> HTTP/1.0 200 OK
        Content-Type: text/html; charset=utf-8
        Allow: POST, OPTIONS
        Access-Control-Allow-Origin: *
        Access-Control-Allow-Headers: Content-Type
        Access-Control-Allow-Methods: DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT
        Content-Length: 0
        Server: Werkzeug/0.9.6 Python/2.7.9
        Date: Sat, 31 Jan 2015 22:25:22 GMT


        $ curl --include -X POST http://127.0.0.1:5000/api/v1/users/create \
            --header Content-Type:application/json \
            --header Origin:www.examplesite.com


        >> HTTP/1.0 200 OK
        Content-Type: application/json
        Content-Length: 21
        Access-Control-Allow-Origin: *
        Server: Werkzeug/0.9.6 Python/2.7.9
        Date: Sat, 31 Jan 2015 22:25:04 GMT

        {
          "success": true
        }

    '''

    return jsonify(success=True)

if __name__ == "__main__":
    app.run(debug=True)
```

CHAPTER 3

Tests

A simple set of tests is included in `test/`. To run, install nose, and simply invoke `nosetests` or `python setup.py test` to exercise the tests.

CHAPTER 4

Contributing

Questions, comments or improvements? Please create an issue on Github, tweet at @corydolphin or send me an email.

CHAPTER 5

Credits

This Flask extension is based upon the Decorator for the HTTP Access Control written by Armin Ronacher.

C