Flask-Cors Documentation

Release 1.7.0

Cory Dolphin

August 22, 2014

Contents

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

CHAPTER 1

Contact

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

CHAPTER 2

Flask-CORS

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

2.1 Installation

Install the extension with using pip, or easy_install.

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

2.2 Usage

This extension enables CORS support either via a decorator, or a Flask extension. 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.

2.2.1 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!"
```

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"
```

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!"
```

2.2.2 Using JSON with CORS

When using JSON cross origin, browsers will issue a pre-flight OPTIONS request for POST requests. In order for browsers to allow POST requests with a JSON content type, you must allow the Content-Type header. The simplest way to do this is to simply set the CORS_HEADERS configuration value on your application: e.g.

```
app.config['CORS_HEADERS'] = 'Content-Type'
```

2.2.3 Full description of options

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 or string*) – The origin, or list of origins to allow requests from.
- **methods** (*list*) – The method or list of methods which the allowed origins are allowed to access.
- **headers** (*list or string*) – The header or list of header field names which can be used when this resource is accessed by allowed origins.
- **expose_headers** – The header or list of headers which are are safe to expose to browsers.
- **supports_credentials** (*bool*) – Allows users to make authenticated requests. If true, injects the *Access-Control-Allow-Credentials* header in responses. Note: According to the W3 spec, this option cannot be used in conjuction with a '*' origin
- **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.
- **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.
- **always_send** (*bool*) – If True, CORS headers are sent even if there is no *Origin* in the request's headers.
- **automatic_options** (*bool*) – If True, CORS headers will be returned for OPTIONS requests. For use with cross domain POST requests which preflight OPTIONS requests, you will need to specifically allow the Content-Type header.

- **vary_header** (*bool*) – If True, the header Vary: Origin will be returned as per suggestion by 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 any Origin other than '*' is returned, informing CDNs and other caches that the CORS headers are dynamic, and cannot be re-used. If False, the Vary header will never be injected or altered.

2.2.4 More examples

2.2.5 A simple, and suggested 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
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(__name__)
# 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/*', 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/wcdolphin/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:
```

```
        $ http get http://127.0.0.1:5000/api/v1/users/
        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(success=True)


@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. The expected result is as follows:

        $ http POST http://127.0.0.1:5000/api/v1/users/create
        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 00:28:26 GMT
        Server: Werkzeug/0.9.4 Python/2.7.8

        {
            "success": true
        }

    '''
    return jsonify(success=True)


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

2.2.6 A more complicated example

If you require advanced configuration and more specific configuration of CORS support for your application, this example provides a useful example of multiple regular expressions and options.

```
"""
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
try:
    # The typical way to import flask-cors
    from flask.ext.cors import CORS, 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 CORS, cross_origin

app = Flask(__name__)


# Set CORS options on app configuration
app.config['CORS_HEADERS'] = "Content-Type"
app.config['CORS_RESOURCES'] = {r"/api/*": {"origins": "*"}}

cors = CORS(app)

## Equivalent to (but using both is not advised)
cors = CORS(app, resources={r"/api/*": {"origins": "*"}},
            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 '''This view is not exposed over CORS.'''


@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:

        $ http get http://127.0.0.1:5000/api/v1/users/
        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(success=True)
```

```
@app.route("/api/v1/users/create", methods=['POST'])
@cross_origin(origins="http://foo.com")
def create_user():
    '''
        This resource both matches the regular expression r'/api/*', and is
        also decorated with the cross_origin decorator. Since The decorator
        actually modifies the view function, and will run before the response
        is touched by the CORS object, the settings for CORS defined in the
        decorator will be used e.g. the allowed origin is only 'http://foo.com',
        rather than the more permissive '*' default.
        Thus, the expected headers are as follows:

        HTTP/1.0 200 OK
        Access-Control-Allow-Headers: Content-Type
        Access-Control-Allow-Origin: http://foo.com
        Content-Length: 21
        Content-Type: application/json
        Date: Sat, 09 Aug 2014 00:32:02 GMT
        Server: Werkzeug/0.9.4 Python/2.7.8
        Vary: Origin

        {
            "success": true
        }
    '''

    return jsonify(success=True)


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

2.2.7 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
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(__name__)


@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:

        $ http get http://127.0.0.1:5000/
        HTTP/1.0 200 OK
        Access-Control-Allow-Origin: *
        Content-Length: 184
        Content-Type: text/html; charset=utf-8
        Date: Sat, 09 Aug 2014 00:35:39 GMT
        Server: Werkzeug/0.9.4 Python/2.7.8

        <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/wcdolphin/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/wcdolphin/flask-cors">Github</a>'''


@app.route("/user/create", methods=['GET', 'POST'])
@cross_origin(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.

        $ http post http://127.0.0.1:5000/user/create
        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 00:38:47 GMT
        Server: Werkzeug/0.9.4 Python/2.7.8

        {
            "success": true
        }

    '''

    return jsonify(success=True)

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