Flask-Cors Documentation

Release 1.6.1

Cory Dolphin

August 22, 2014

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 flask-cors
```

CHAPTER 2

Usage

This extension 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.

2.1 Simple Usage

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

2.2 Using JSON with Cross Origin

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.

```
@app.route("/user/create", methods=['GET','POST'])
@cross_origin(headers=['Content-Type']) # Send Access-Control-Allow-Headers
def cross_origin_json_post():
  return jsonify(success=True)
```

2.3 Application-wide settings

Alternatively, you can set any of these options in an app's config
object. Setting these at the application level effectively changes the
default value for your application, while still allowing you to

override | it on a per-resource basis.

The application-wide configuration options are creatively prefixed

with CORS_ | e.g.

- CORS_ORIGINS
- CORS_METHODS
- CORS_HEADERS
- CORS_EXPOSE_HEADERS
- CORS_ALWAYS_SEND
- CORS_MAX_AGE
- CORS_SEND_WILDCARD
- CORS_ALWAYS_SEND
- CORS_AUTOMATIC_OPTIONS

```
app.config['CORS_ORIGINS'] = ['https://foo.com', 'http://www.bar.com']
app.config['CORS_HEADERS'] = ['Content-Type']

# Will return CORS headers for origins 'https://foo.com'and 'http://www.bar.com'
# and an Access-Control-Allow-Headers of 'Content-Type'
"""
Will return CORS headers for origins 'https://foo.com'and
'http://www.bar.com' and an Access-Control-Allow-Headers of 'Content-Type'
E.G. Testing with httpie
      ~  http GET http://127.0.0.1:5000/
    HTTP/1.0 200 OK
    Access-Control-Allow-Headers: Content-Type
    Access-Control-Allow-Origin: https://foo.com, http://www.bar.com
    Content-Length: 26
    Content-Type: text/html; charset=utf-8
    Date: Tue, 22 Jul 2014 23:55:53 GMT
    Server: Werkzeug/0.9.4 Python/2.7.8

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


"""
Will return CORS headers for 'google.com' and  Access-Control-Allow-Headers of
'Content-Type'.
E.G. Testing with httpie
       ~  http GET http://127.0.0.1:5000/special
    HTTP/1.0 200 OK
    Access-Control-Allow-Headers: Content-Type
    Access-Control-Allow-Origin: http://google.com
    Content-Length: 33
    Content-Type: text/html; charset=utf-8
    Date: Tue, 22 Jul 2014 23:55:29 GMT
    Server: Werkzeug/0.9.4 Python/2.7.8
    Hello, google-cross-origin-world!
"""
@app.route("/special")
@cross_origin(origins="http://google.com")
def helloGoogle():
  return "Hello, google-cross-origin-world!"
```

For a full list of options, please see the full documentation

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 @wcdolphin or send me an email.

4.1 Full list 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** – 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 Fals, the Vary header will never be injected or altered.