Flask-Dropzone Documentation

Release 1.5.1

Grey Li

May 01, 2021

Contents

Upload files in Flask application with Dropzone.js.

NOTICE: This extension is built for simple usage, if you need more flexibility, please use Dropzone.js directly.

CHAPTER 1

Contents

1.1 Basic Usage

1.1.1 Installation

```
$ pip install flask-dropzone
```

1.1.2 Initialization

Initialize the extension:

```
from flask_dropzone import Dropzone

app = Flask(__name__)
dropzone = Dropzone(app)
```

This extension also supports the Flask application factory pattern by allowing you to create a Dropzone object and then separately initialize it for an app:

```
dropzone = Dropzone()

def create_app(config):
    app = Flask(__name__)
    ...
    dropzone.init_app(app)
    ...
    return app
```

1.1.3 Include Dropzone.js Resources

In addition to manage and load resources by yourself (recommended), you can also use this methods to load resources:

```
<head>
{{ dropzone.load_css() }}
</head>
<body>
...
{{ dropzone.load_js() }}
</body>
```

Tip: There is a `dropzone.load()` method that was a combination of `dropzone.load_css()` and `dropzone.load_js()`, but we recommend not to use this method for page load performance consideration. Also, `dropzone.load()` will be removed in the near future.

You can assign the version of Dropzone.js through `version` argument, the default value is `5.2.0`. And, you can pass `css_url` and `js_url` separately to customize resources URL.

1.1.4 Create a Drop Zone

Creating a Drop Zone with `create()` and use `config()` to make the configuration come into effect:

```
{{ dropzone.create(action='the_url_or_endpoint_which_handle_uploads') }}
...
{{ dropzone.config() }}
</body>
```

Remember to edit the `action` to the URL or endpoint which handles the uploads, for example `dropzone.create(action='upload_view')` or `dropzone.create(action=url_for('upload_view')')`.

The default ID of the dropzone form element is *myDropzone*, usually you don't need to change it. If you have specific need, for example, you want to have multiple dropzones on one page, you can use the `id` parameter to assign the id:

```
{{ dropzone.create(id='foo') }}
{{ dropzone.create(id='bar') }}
...
{{ dropzone.config(id='foo') }}
{{ dropzone.config(id='bar') }}
</body>
```

Notice that the same id must passed both in `dropzone.create()` and `dropzone.config()`.

1.1.5 Beautify Dropzone

Style it according to your preferences through `dropzone.style()` method:

```
<head>
{{ dropzone.load_css() }}
{{ dropzone.style('border: 2px dashed #0087F7; margin: 10%; min-height: 400px;') }}
</head>
```

1.1.6 Save Uploads with Flask

When the file was dropped on drop zone, you can get the uploaded file in `request.files`, just pass upload input's name attribute (default to `file`).

```
import os

from flask import Flask, request
from flask_dropzone import Dropzone

app = Flask(__name__)

dropzone = Dropzone(app)

@app.route('/uploads', methods=['GET', 'POST'])
def upload():

    if request.method == 'POST':
        f = request.files.get('file')
        f.save(os.path.join('the/path/to/save', f.filename))

    return 'upload template'
```

Tip: See `examples/basic` for more detail.

1.2 Configuration

1.2.1 Register Configuration

Except `DROPZONE_SERVE_LOCAL`, when you use other configuration variable, you have to call `dropzone.config()` in template to make them register with Dropzone:

```
<body>
    ...
    {{ dropzone.config() }}
</body>
```

Tip: Call this method after `dropzone.load_js()` or `<script>` that include Dropzonejs.

1.2.2 Available Configuration

The supported list of config options is shown below:

Name	Default Value	Info
DROPZONE_SERVE_LOCAL	`False`	Default to retrieve dropzone.js from CDN
DROPZONE_MAX_FILE_SIZE	`3`	Max allowed file size. unit: MB
DROPZONE_INPUT_NAME	`file`	The name attribute in `<input>` (i.e. `<input type="file" name="file">`)
DROPZONE_ALLOWED_FILE_CUSTOM	`False`	See detail below
DROPZONE_ALLOWED_FILE_TYPE	`'default'`	See detail below
DROPZONE_MAX_FILES	`'null'`	The max files user can upload once
DROPZONE_DEFAULT_MESSAGE	`"Drop files here to upload"`	Message displayed on drop area, you can write HTML here (e.g. `Drop files here Or <button type="button">Click to Upload</button>`)
DROPZONE_INVALID_FILE_TYPE	`"You can't upload files of this type."`	Error message
DROPZONE_FILE_TOO_BIG	`"File is too big {{filesize}}. Max filesize: {{maxFilesize}}MiB. "`	Error message
DROPZONE_SERVER_ERROR	`'"Server error: {{statusCode}}"'`	Error message
DROPZONE_BROWSER_UNSUPPORTED	`"Your browser does not support drag'n'drop file uploads."`	Error message
DROPZONE_MAX_FILE_EXCEED	`"You can't upload any more files."`	Error message
DROPZONE_UPLOAD_MULTIPLE	`False`	Whether to send multiple files in one request.
DROPZONE_PARALLEL_UPLOADS	`2`	How many uploads will handled in per request when `DROPZONE_UPLOAD_MULTIPLE` set to `True`.
DROPZONE_REDIRECT_VIEW	`None`	The view to redirect when upload was completed. If you want pass an URL, usually when your view accepts variable, you can pass it with `redirect_url` keyword in template: `{{ dropzone.config(redirect_url=url_for('endpoint', foo=bar)) }}`.
DROPZONE_ENABLE_CSRF	`False`	Enable CSRF protect, see detail below
DROPZONE_TIMEOUT	`None`	The timeout to cancel upload request in millisecond, default to `30000` (30 second). Set a large number if you need to upload large file.

1.2.3 File Type Filter

Just set `DROPZONE_ALLOWED_FILE_TYPE` to one of `default`, `image`, `audio`, `video`, `text`, `app`, for example:

```
app.config['DROPZONE_ALLOWED_FILE_TYPE'] = 'image'
```

If you want to set the allowed file type by yourself, you need to set `DROPZONE_ALLOWED_FILE_CUSTOM` to `True`, then add mime type or file extensions to `DROPZONE_ALLOWED_FILE_TYPE`, such as:

```
app.config['DROPZONE_ALLOWED_FILE_CUSTOM'] = True
app.config['DROPZONE_ALLOWED_FILE_TYPE'] = 'image/*, .pdf, .txt'
```

Consult the dropzone.js documentation for details on these options.

1.2.4 Custom Configuration String

Sometimes you may need more flexible, you can use `custom_init``and ``custom_options` to pass custom JavaScript code:

```
{{ dropzone.config(custom_init='dz = this;document.getElementById("upload-btn").
↪addEventListener("click", function handler(e) {dz.processQueue();});',
                   custom_options='autoProcessQueue: false, addRemoveLinks: true,␣
↪parallelUploads: 20,') }}
```

The code pass with `custom_init` will into `init: function() {}`, the code pass with `custom_options` will into `Dropzone.options.myDropzone = {}`. See the full list of available configuration settings on Dropzone documentation.

1.2.5 Overwriting Global Configuration

Sometimes you may want to use different configuration for multiple drop area on different pages, in this case, you can pass the specific keyword arguments into `dropzone.config()` directly.

The keyword arguments should mapping the corresponding configration variable in this way:

- DROPZONE_DEFAULT_MESSAGE –> default_message
- DROPZONE_TIMEOUT –> timeout
- DROPZONE_ALLOWED_FILE_TYPE –> allowed_file_type
- etc

example:

```
{{ dropzone.config(max_files=10, timeout=10000, default_message='Drop here!') }}
```

In the end, the keyword argument you pass will overwrite the corresponding configurations.

1.3 Advanced Usage

1.3.1 Parallel Uploads

If you set `DROPZONE_UPLOAD_MULTIPLE` as True, then you need to save multiple uploads in single request.

However, you can't get a list of file with `request.files.getlist('file')`. When you enable parallel upload, Dropzone.js will append a index number after each files, for example: `file[2]`, `file[1]`, `file[0]`. So, you have to save files like this:

```
for key, f in request.files.items():
    if key.startswith('file'):
        f.save(os.path.join('the/path/to/save', f.filename))
```

Here is the full example:

```
...
app.config['DROPZONE_UPLOAD_MULTIPLE'] = True  # enable parallel upload
app.config['DROPZONE_PARALLEL_UPLOADS'] = 3  # handle 3 file per request

@app.route('/upload', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        for key, f in request.files.items():
            if key.startswith('file'):
                f.save(os.path.join('the/path/to/save', f.filename))

   return 'upload template'
```

Tip: See `examples/parallel-upload` for more detail.

1.3.2 CSRF Protect

The CSRF Protect feature was provided by Flask-WTF's `CSRFProtect` extension, so you have to install Flask-WTF first:

```
$ pip install flask-wtf
```

Then initialize the CSRFProtect:

```
from flask_wtf.csrf import CSRFProtect

app = Flask(__name__)

# the secret key used to generate CSRF token
app.config['SECRET_KEY'] = 'dev key'
...
# enable CSRF protection
app.config['DROPZONE_ENABLE_CSRF'] = True

csrf = CSRFProtect(app)
```

Make sure to set the secret key and set `DROPZONE_ENABLE_CSRF` to True. Now all the upload request will be protected!

We prefer to handle the CSRF error manually, because the error response's body will be displayed as tooltip below the file thumbnail.

```
from flask_wtf.csrf import CSRFProtect, CSRFError
...

# handle CSRF error
@app.errorhandler(CSRFError)
```

(continues on next page)

(continued from previous page)

```
def csrf_error(e):
    return e.description, 400
```

Here I use the `e.description` as error message, it's provided by CSRFProtect, one of `The CSRF token is missing` and `The CSRF token is invaild`.

Try the demo application in `examples/csrf` and see CSRFProtect's documentation for more details.

1.3.3 Content Security Policy

If you like to use your web application under a strict Content Security Policy (CSP), just embedding JavaScript code via `{{ dropzone.config() }}` into a template will not work. You could move the configuration code into a separate JavaScript file and reference this resource from your HTML page. However, when you like to enable a CSRF protection as well, you need to handle the CSRF token and the CSP nonce value. The simple solution is to embedd the configuration code into the HTML page and pass a `nonce` value for CSP as shown below:

```
import base64
import os

default_http_header = {'Content-Security-Policy' :
  f"default-src 'self'; script-src 'self' 'nonce-{nonce}'"

nonce = base64.b64encode(os.urandom(64)).decode('utf8')
render_template('template.tmpl', nonce = nonce), 200, default_http_header
```

```
{{ dropzone.config(nonce=nonce) }}
```

1.3.4 Server Side Validation

Although Dropzone.js can handle client side validation for uploads, but you still need to setup server side validation for security conern. Just do what you normally do (extension check, size check etc.), the only thing you should remember is to return plain text error message as response body when something was wrong. Fox example, if we only want user to upload file with `.png` extension, we can do the validation like this:

```
@app.route('/', methods=['POST', 'GET'])
def upload():
    if request.method == 'POST':
        f = request.files.get('file')
        if f.filename.split('.')[1] != 'png':
            return 'PNG only!', 400  # return the error message, with a proper 4XX␣
↪code
        f.save(os.path.join('the/path/to/save', f.filename))
    return render_template('index.html')
```

The error message will be displayed when you hover the thumbnail for upload file:

1.4 Try Examples

Open a terminal, type the commands below one by one:

Fig. 1: error message

```
$ git clone https://github.com/greyli/flask-dropzone
$ cd flask-dropzone/examples
$ pip install -r requirements.txt
$ python basic/app.py
```

Then go to http://127.0.0.1:5000 with your favourite browser.

Aside from the basic example, there are a couple of additional examples:

- examples/click-upload
- examples/complete-redirect
- examples/csrf
- examples/custom-options
- examples/in-form
- examples/large-file
- examples/parallel-upload
- examples/multiple-dropzone

CHAPTER 2

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API Reference

2.1.1 Dropzone Object in Template

class flask_dropzone.**_Dropzone**

static config(*redirect_url=None*, *custom_init=''*, *custom_options=''*, *nonce=None*, *id=None*, ***kwargs*)

Initialize dropzone configuration.

Changed in version 1.5.4: Added `id` parameter.

New in version 1.4.4.

Parameters

- **redirect_url** – The URL to redirect when upload complete.
- **custom_init** – Custom javascript code in `init: function() {}`.
- **custom_options** – Custom javascript code in `Dropzone.options.myDropzone = {}`.
- **nonce** – Pass a nonce value that is newhen embedding the JavaScript code into a Content Security Policy protected web page.
- **id** – The id of the dropzone element, it must matches the `id` argument passed to `dropzone.create()` if provided.
- ****kwargs** – Mirror configuration variable, lowercase and without prefix. For example, `DROPZONE_UPLOAD_MULTIPLE` becomes `upload_multiple` here.

static create(*action=''*, *csrf=False*, *action_view=''*, *id=None*, ***kwargs*)
Create a Dropzone form with given action.

Changed in version 1.4.2: Added `csrf` parameter to enable CSRF protect.

Changed in version 1.4.3: Added `action` parameter to replace `action_view`, `action_view` was deprecated now.

Changed in version 1.5.0: If `DROPZONE_IN_FORM` set to `True`, create `<div>` instead of `<form>`.

Changed in version 1.5.4: `csrf` was deprecated now.

Changed in version 1.5.4: Added `id` parameter.

Parameters

- **action** – The action attribute in `<form>`, pass the url which handle uploads.
- **csrf** – Enable CSRF protect or not, same with `DROPZONE_ENABLE_CSRF`, deprecated since 1.5.4.
- **action_view** – The view which handle the post data, deprecated since 1.4.2.
- **id** – The id of the dropzone element, it must matches the `id` argument passed to `dropzone.config()` if provided.

static load(*js_url=''*, *css_url=''*, *version='5.2.0'*)
Load Dropzone resources with given version and init dropzone configuration.

Changed in version 1.4.3: Added `js_url` and `css_url` parameters to pass custom resource URL.

Changed in version 1.4.4: This method was deprecated due to inflexible. Now it's divided into three methods: 1. Use `load_css()` to load css resources. 2. Use `load_js()` to load js resources. 3. Use `config()` to configure Dropzone.

Parameters

- **js_url** – The JavaScript url for Dropzone.js.
- **css_url** – The CSS url for Dropzone.js.
- **version** – The version of Dropzone.js.

static load_css(*css_url=None*, *version='5.2.0'*)
Load Dropzone's css resources with given version.

New in version 1.4.4.

Parameters

- **css_url** – The CSS url for Dropzone.js.
- **version** – The version of Dropzone.js.

static load_js(*js_url=None*, *version='5.2.0'*)
Load Dropzone's js resources with given version.

New in version 1.4.4.

Parameters

- **js_url** – The JS url for Dropzone.js.
- **version** – The version of Dropzone.js.

static style(*css*)
Add css to dropzone.

Parameters **css** – style sheet code.

2.1.2 Utils

flask_dropzone.utils.**get_url**(*endpoint_or_url*, ***kwargs*)

flask_dropzone.utils.**random_filename**(*old_filename*)

CHAPTER 3

Changelog

3.1 Changelog

3.1.1 2.0.0

released date: –

WARNING: **New major upstream release (backwards incompatible!).**

- Remove `dropzone.load()` method.
- Added more options to customize messages.
- Drop Python 2 support.

3.1.2 1.6.0

released date: 2021/5/1

- Add a `id` parameter for `dropzone.create()` and `dropzone.config()` to support

customize element id and putting multiple dropzones in one page.

3.1.3 1.5.4

released date: 2019/8/4

- Fix CSRF protect bug when in form (#29)

3.1.4 1.5.3

released date: 2018/8/24

- Built-in resources behaviour will not based on `FLASK_ENV`.
- Add support to pass configuration variable directly with `dropzone.config()`, see documentation for more details.

3.1.5 1.5.2

released date: 2018/8/7

- Add a proper documentation.
- Fix KeyError Exception if ENV isn't defined.

3.1.6 1.5.1

released date: 2018/7/21

- Change CDN provider to jsDelivr.
- Built-in resources will be used when `FLASK_ENV` set to `development`.

3.1.7 1.5.0

released date: 2018/7/20

- `action` in `dropzone.create()` can be URL or endpoint, `action_view` was deprecated.
- Add support to upload all dropped files when specific button (`name="upload"`) was clicked.
- Add configuration variable `DROPZONE_UPLOAD_ON_CLICK`, `DROPZONE_UPLOAD_ACTION`, `DROPZONE_UPLOAD_BTN_ID`.
- Add configuration variable `DROPZONE_IN_FORM`, `DROPZONE_UPLOAD_ACTION` to support create dropzone inside `<form>`.
- Add configuration variable `DROPZONE_TIMEOUT`.
- Add `custom_init` and `custom_options` parameters in `dropzone.config()` to support pass custom JavaScript.

3.1.8 1.4.6

released date: 2018/6/8

- Change built-in resource's url path to `dropzone/static/...` to prevent conflict with user's static path.

3.1.9 1.4.4

released date: 2018/5/28

- `dropzone.load()` method was deprecated due to inflexible. Now it's divided into three methods: * Use `load_css()` to load css resources. * Use `load_js()` to load js resources. * Use `config()` to configure Dropzone. * Besides, we recommend user to manage the resouces manually.
- Add basic unit tests.

3.1.10 1.4.3

released date: 2018/3/23

- Add support to use custom resources with `js_url` and `css_url` param in `load()`.
- Fix built-in static bug (#11).
- Use package instead of module.

3.1.11 1.4.2

released date: 2018/2/17

- Add support to integrate with CSRFProtect (enabled via `DROPZONE_ENABLE_CSRF` or `csrf` flag in `dropzone.create()`).
- Fix bug: `False` in JavaScript.
- Bump built-in resource's version to 5.2.0
- Add `action` argument in `dropzone.create()`. For example, `dropzone.create(action=url_for('upload'))`.

3.1.12 1.4.1

- New configuration options: `DROPZONE_UPLOAD_MULTIPLE`, `DROPZONE_PARALLEL_UPLOADS`, `DROPZONE_REDIRECT_VIEW`.
- Fix local static files bug.
- Add support for automatic redirection when upload was conmplete.

3.1.13 1.4

WARNING: **New major upstream release (backwards incompatible!).**

- Method `include_dropzone()` rename to `load()`.
- Add a `create()` method to create dropzone form.
- Add a `style()` method to add style to upload area.
- Use `action_view` argument (in `create()`) to set action url.
- Dropzonejs version increase to 5.1.1.
- PEP8 and bug fix.

3.1.14 1.3

- Documentation fix.

3.1.15 1.2

- Upload address fix.
- Delete useless code.

3.1.16 1.1

- Add more configuration options.
- Support local resource serve.
- Add basic documentation.

3.1.17 1.0

- Init release.

CHAPTER 4

Development

We welcome all kinds of contributions. You can run test like this:

```
$ python setup.py test
```

CHAPTER 5

Authors

Maintainer: Grey Li

See also the list of contributors who participated in this project.

CHAPTER 6

License

This project is licensed under the MIT License (see the `LICENSE` file for details).

Python Module Index

f

Index

Symbols

C

F

G

L

R

S