Flask-Uploads Documentation

Release 0.1.1

Matthew "LeafStorm" Frazier

Nov 02, 2017

Contents

Flask-Uploads allows your application to flexibly and efficiently handle file uploading and serving the uploaded files. You can create different sets of uploads - one for document attachments, one for photos, etc. - and the application can be configured to save them all in different places and to generate different URLs for them.

CHAPTER 1

Configuration

If you're just deploying an application that uses Flask-Uploads, you can customize its behavior extensively from the application's configuration. Check the application's documentation or source code to see how it loads its configuration.

The settings below apply for a single set of uploads, replacing `FILES` with the name of the set (i.e. `PHOTOS`, `ATTACHMENTS`):

`UPLOADED_FILES_DEST` This indicates the directory uploaded files will be saved to.

`UPLOADED_FILES_URL` If you have a server set up to serve the files in this set, this should be the URL they are publicly accessible from. Include the trailing slash.

`UPLOADED_FILES_ALLOW` This lets you allow file extensions not allowed by the upload set in the code.

`UPLOADED_FILES_DENY` This lets you deny file extensions allowed by the upload set in the code.

To save on configuration time, there are two settings you can provide that apply as "defaults" if you don't provide the proper settings otherwise.

`UPLOADS_DEFAULT_DEST` If you set this, then if an upload set's destination isn't otherwise declared, then its uploads will be stored in a subdirectory of this directory. For example, if you set this to `/var/uploads`, then a set named photos will store its uploads in `/var/uploads/photos`.

`UPLOADS_DEFAULT_URL` If you have a server set up to serve from `UPLOADS_DEFAULT_DEST`, then set the server's base URL here. Continuing the example above, if `/var/uploads` is accessible from `http://localhost:5001`, then you would set this to `http://localhost:5001/` and URLs for the photos set would start with `http://localhost:5001/photos`. Include the trailing slash.

However, you don't have to set any of the `_URL` settings - if you don't, then they will be served internally by Flask. They are just there so if you have heavy upload traffic, you can have a faster production server like Nginx or Lighttpd serve the uploads.

If you are running Flask 0.6 or greater, or the application uses `patch_request_class(app, None)`, then you can set `MAX_CONTENT_LENGTH` to limit the size of uploaded files.

CHAPTER 2

Upload Sets

An "upload set" is a single collection of files. You just declare them in the code:

```
photos = UploadSet('photos', IMAGES)
```

And then you can use the `save` method to save uploaded files and `path` and `url` to access them. For example:

```
@app.route('/upload', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST' and 'photo' in request.files:
        filename = photos.save(request.files['photo'])
        rec = Photo(filename=filename, user=g.user.id)
        rec.store()
        flash("Photo saved.")
        return redirect(url_for('show', id=rec.id))
    return render_template('upload.html')

@app.route('/photo/<id>')
def show(id):
    photo = Photo.load(id)
    if photo is None:
        abort(404)
    url = photos.url(photo.filename)
    return render_template('show.html', url=url, photo=photo)
```

If you have a "default location" for storing uploads - for example, if your app has an "instance" directory like Zine and uploads should be saved to the instance directory's `uploads` folder - you can pass a `default_dest` callable to the set constructor. It takes the application as its argument. For example:

```
media = UploadSet('media', default_dest=lambda app: app.instance_path)
```

This won't prevent a different destination from being set in the config, though. It's just to save your users a little configuration time.

CHAPTER 3

App Configuration

An upload set's configuration is stored on an app. That way, you can have upload sets being used by multiple apps at once. You use the `configure_uploads` function to load the configuration for the upload sets. You pass in the app and all of the upload sets you want configured. Calling `configure_uploads` more than once is safe.

```
configure_uploads(app, (photos, media))
```

If your app has a factory function, that is a good place to call this function.

By default, though, Flask doesn't put any limits on the size of the uploaded data. To protect your application, you can use `patch_request_class`. If you call it with `None` as the second parameter, it will use the `MAX_CONTENT_LENGTH` setting to determine how large the upload can be.

```
patch_request_class(app, None)
```

You can also call it with a number to set an absolute limit, but that only exists for backwards compatibility reasons and is not recommended for production use. In addition, it's not necessary for Flask 0.6 or greater, so if your application is only intended to run on Flask 0.6, you don't need it.

CHAPTER 4

File Upload Forms

To actually upload the files, you need to properly set up your form. A form that uploads files needs to have its method set to POST and its enctype set to `multipart/form-data`. If it's set to GET, it won't work at all, and if you don't set the enctype, only the filename will be transferred.

The field itself should be an `<input type=file>`.

```
<form method=POST enctype=multipart/form-data action="{{ url_for('upload') }}">
    ...
    <input type=file name=photo>
    ...
</form>
```

CHAPTER 5

API Documentation

This documentation is generated directly from the source code.

5.1 Upload Sets

5.2 Application Setup

5.3 Extension Constants

These are some default sets of extensions you can pass to the `UploadSet` constructor.

5.4 Testing Utilities

CHAPTER 6

Backwards Compatibility

6.1 Version 0.1.3

- The `_uploads` module/blueprint will not be registered if it is not needed to serve uploads.

6.2 Version 0.1.1

- `patch_request_class` now changes `max_content_length` instead of `max_form_memory_size`.