Flask-Store

Release 0.0.2

July 21, 2014

Contents

`Flask-Store` is a Flask Extension designed to provide easy file upload handling in the same vien as Django-Storages, allowing developers to user custom storage backends or one of the provided storage backends.

Warning: This Flask Extenstion is under heavy development. It is likely API's will change without warning.

CHAPTER 1

Included Providers

- Local File System
- AWS Simple Storage Service (S3)

CHAPTER 2

Usage Documentation

2.1 Installation

Simply grab it from PyPI:

```
pip install Flask-Store
```

2.2 Quick Start

Getting up and running with Flask-Store is pretty easy. By default Flask-Store will use local file system storage to store your files. All you need to do is to tell it where you want your uploaded files to live.

2.2.1 Step 1: Integration

First lets initialise the Flask-Store extension with our Flask application object.

```
from flask import Flask
from flask.ext.store import Store

app = Flask(__name__)
store = Store(app)

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

That is all there is to it. If you use an application factory then you can use `flask_store.Store.init_app()` method instead:

```
from flask import Flask
from flask.ext.store import Store

store = Store()

def create_app():
    app = Flask(__name__)
    store.init_app(app)

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

2.2.2 Step 2: Configuration

So all we need to do now is tell Flask-Store where to save files once they have been uploaded. For asolute url generation we also need to tell Flask-Store about the domain where the files can accessed.

To do this we just need to set a configuration variable called `STORE_PATH` and `STORE_DOMAIN`.

For brevity we will not show the application factory way because its pretty much identical.

```
from flask import Flask
from flask.ext.store import Store

app = Flask(__name__)
app.config['STORE_DOMAIN'] = 'http://127.0.0.1:5000'
app.config['STORE_PATH'] = '/some/path/to/somewhere'
store = Store(app)

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

Now when Flask-Store saves a file it will be located here: `/some/path/to/somewhere`.

2.2.3 Step 3: Add a route

Now we just need to save a file. We just need a route which gets a file from the request object and send it to our Flask-Store Provider (by default local Storage) to save it.

Note: It is important to note the Flask-Store makes no attempt to validate your file size, extensions or what not, it just does one thing and that is save files somewhere. So if you need validation you should use something like `WTForms` to validate incoming data from the user.

```
from flask import Flask, request
from flask.ext.store import Store

app = Flask(__name__)
app.config['STORE_DOMAIN'] = 'http://127.0.0.1:5000'
app.config['STORE_PATH'] = '/some/path/to/somewhere'
store = Store(app)

@app.route('/upload', methods=['POST', ])
def upload():
    file = request.files.get('afile')
    provider = store.Provider()
    f = provider.save(file)

    return f.absolute_url()

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

Now if we were to `curl` a file to our upload route we should get a url back which tells how we can access it.

```
curl -i -F afile=@localfile.jpg http://127.0.0.1:5000/upload
```

We should get back something like:

```
HTTP/1.1 100 Continue

HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 44
Server: Werkzeug/0.9.6 Python/2.7.5
Date: Thu, 17 Jul 2014 11:32:02 GMT

http://127.0.0.1:5000/flaskstore/localfile.jpg%
```

Now if you went to `http://127.0.0.1:5000/flaskstore/localfile.jpg` in your browser you should see the image you uploaded. That is because Flask-Store automatically registers a route for serving files.

Note: By the way, if you don't like the url you can change it by setting `STORE_URL_PREFIX` in your application configuration.

2.2.4 Step 4: There is no Step 4

Have a beer (or alcoholic beverage (or not) of your choice), that was exhausting.

2.3 Local Store

Note: This document assumes you have already read the *Quick Start* guide.

As we discussed in the *Quick Start* guide Flask-Store uses its `flask_store.stores.local.LocalStore` as its default provider and here we will discuss some of the more advanced concepts of this store provider.

2.3.1 Enable

This is the default provider but if you wish to be explicit (+1) then simply set the following in your application configuration:

```
STORE_PROVIDER='flask_store.stores.local.LocalStore'
```

2.3.2 Configuration

The following configuration variables are available for you to customise.

Name	Example Value
`STORE_PATH`	`/somewhere/on/disk`
This tells Flask-Store where to save uploaded files too. For this provider it must be an absolute path to a location on disk you have per	
`STORE_URL_PREFIX`	`/uploads`
Used to generate the URL for the uploaded file. The `LocalStore` will automatically register a route with your Flask application so	

2.4 S3 Store

Note: This document assumes you have already read the *Quick Start* guide.

The S3 Store allows you to forward your uploaded files up to an AWS Simple Storage Service (S3) bucket. This takes the problem of storing large numbers of files away from you onto Amazon.

Note: Amazon's `boto` is required. Boto is not included as a install requirement for Flask-Store as not everyone will want to use the S3 provider. To install just run:

```
pip install boto
```

2.4.1 Enable

To use this provider simply set the following in your application configuration:

```
STORE_PROVIDER='flask_store.stores.s3.S3Store'
```

2.4.2 Configuration

The following configuration variables are availible to you.

Name	Example Value
`STORE_PATH`	`/some/place/in/bucket`
For the `S3Store` is basically your key name prefix rather than an actual location. So for the example value above the key for a file m	
`STORE_DOMAIN`	`https://bucket.s3.amazonaws.com`
Your S3 bucket domain, this is used to generate an absolute url.	
`STORE_S3_REGION`	`us-east-1`
The region in which your bucket lives	
`STORE_S3_BUCKET`	`your.bucket.name`
The name of the S3 bucket to upload files too	
`STORE_S3_ACCESS_KEY`	`ABCDEFG12345`
Your AWS access key which has permission to upload files to the `STORE_S3_BUCKET`.	
`STORE_S3_SECRET_KEY`	`ABCDEFG12345`
Your AWS access secret key	

2.5 S3 Gevent Store

Note: This document assumes you have already read the *Quick Start* guide.

The `flask_store.stores.s3.S3GeventStore` allows you to run the upload to S3 process in a Gevent Greenlet process. This allows your webserver to send a response back to the client whilst the upload to S3 happends in the background.

Obviously this means that when the request has finished the upload may not have finished and the key not exist in the bucket. You will need to build your application around this.

Note: The `gevent` package is required. Gevent is not included as a install requirement for Flask-Store as not everyone will want to use the S3 Gevent provider. To install just run:

```
pip install gevent
```

2.5.1 Enable

To use this provider simply set the following in your application configuration:

```
STORE_PROVIDER='flask_store.stores.s3.S3GeventStore'
```

2.5.2 Configuration

Note: This is a sub class of `flask_store.stores.s3.S3Store` and therefore all the same confiuration options apply.

CHAPTER 3

Reference

3.1 API Reference

3.1.1 flask_store

Adds simple file handling for different providers to your application. Provides the following providers out of the box:

- Local file storeage
- Amazon Simple File Storage (requires `boto` to be installed)

class `flask_store.`**`Store`**(*app=None*)

Flask-Store integration into Flask applications. Flask-Store can be integrated in two different ways depending on how you have setup your Flask application.

You can bind to a specific flask application:

```
app = Flask(__name__)
store = Store(app)
```

Or if you use an application factory you can use `flask_store.Store.init_app()`:

```
store = Store()
def create_app():
    app = Flask(__name__)
    store.init_app(app)
    return app
```

`check_config`(*app*)

Checks the required application configuration variables are set in the flask application.

Parameters **app** (*flask.app.Flask*) – Flask application instance

Raises `NotConfiguredError` – In the event a required config parameter is required by the Store.

`init_app`(*app*)

Sets up application default confugration options and sets a `Provider` property which can be used to access the default provider class which handles the saving of files.

Parameters **app** (*flask.app.Flask*) – Flask application instance

`provider`(*app*)

Fetches the provider class as defined by the application configuration.

Parameters **app** (*flask.app.Flask*) – Flask application instance

Raises `ImportError` – If the class or module cannot be imported

Returns The provider class

Return type class

register_route(*app*)
Registers a default route for serving uploaded assets via Flask-Store, this is based on the absolute and relative paths defined in the app configuration.

Parameters **app** (*flask.app.Flask*) – Flask application instance

set_provider_defaults(*app*)
If the provider has a `app_defaults` static method then this simply calls that method. This will set sensible application configuration options for the provider.

Parameters **app** (*flask.app.Flask*) – Flask application instance

class flask_store.**StoreState**(*store*, *app*)
Stores the state of Flask-Store from application init.

flask_store.**store_provider**()
Returns the default provider class as defined in the application configuration.

Returns The provider class

Return type class

3.1.2 flask_store.exceptions

Custom Flask-Store exception classes.

exception flask_store.exceptions.**NotConfiguredError**
Raise this exception in the event the flask application has not been configured properly.

3.1.3 flask_store.files

class flask_store.files.**StoreFile**(*filename*, *destination=None*)
An Ambassador class for the provider for a specific file. Each method basically proxies to methods on the provider.

absolute_path()
Returns the absollute file path to the file.

Returns Absolute file path

Return type str

absolute_url()
Absolute url contains a domain if it is set in the configuration, the url predix, destination and the actual file name.

Returns Full absolute URL to file

Return type str

relative_path()
Returns the relative path to the file, so minus the base path but still includes the destination if it is set.

Returns Relative path to file

Return type str

relative_url()
Returns the relative URL, basically minus the domain.

Returns Realtive URL to file

Return type str

3.1.4 flask_store.utils

flask_store.utils.**path_to_uri**(*path*)
Swaps for / Other stuff will happen here in the future.

3.1.5 flask_store.stores

Base store functionality and classes.

class flask_store.stores.**BaseStore**(*destination=None*)
Base file storage class all storage providers should inherit from. This class provides some of the base functionality for all providers. Override as required.

exists(**args*, ***kwargs*)
Placeholder "exists" method. This should be overridden by custom providers and return a boolean depending on if the file exists of not for the provider.

Raises NotImplementedError – If the "exists" method has not been implemented

join(**args*, ***kwargs*)
Each provider needs to implement how to safely join parts of a path together to result in a path which can be used for the provider.

Raises NotImplementedError – If the "join" method has not been implemented

register_route = False
By default Stores do not require a route to be registered

safe_filename(*filename*)
If the file already exists the file will be renamed to contain a short url safe UUID. This will avoid overwtites.

Parameters **filename** (*str*) – A filename to check if it exists

Returns A safe filenaem to use when writting the file

Return type str

save(**args*, ***kwargs*)
Placeholder "sabe" method. This should be overridden by custom providers and save the file object to the provider.

Raises NotImplementedError – If the "save" method has not been implemented

url_join(**parts*)
Safe url part joining.

Parameters ***parts** – List of parts to join together

Returns Joined url parts

Return type str

3.1.6 flask_store.stores.local

Local file storage for your Flask application.

Example

```
from flask import Flask, request
from flask.ext.store import Provider, Store
from wtforms import Form
from wtforms.fields import FileField

class FooForm(Form):
    foo = FileField('foo')

app = Flask(__app__)
app.config['STORE_PATH'] = '/some/file/path'

store = Store(app)

@app,route('/upload')
def upload():
    form = FooForm()
    form.validate_on_submit()

    if not form.errors:
        provider = store.Provider()
        provider.save(request.files.get('foo'))
```

class `flask_store.stores.local.`**LocalStore**(*destination=None*)

The default provider for Flask-Store. Handles saving files onto the local file system.

static app_defaults(*app*)

Sets sensible application configuration settings for this provider.

Parameters **app** (*flask.app.Flask*) – Flask application at init

exists(*filename*)

Returns boolean of the provided filename exists at the compiled absolute path.

Parameters **name** (*str*) – Filename to check its existence

Returns Whether the file exists on the file system

Return type bool

join(**parts*)

Joins paths together in a safe manor.

Returns Joined paths

Return type str

register_route = True

Ensure a route is registered for serving files

save(*file*)

Save the file on the local file system. Simply builds the paths and calls `werkzeug.datastructures.FileStorage.save()` on the file object.

Parameters **file** (*werkzeug.datastructures.FileStorage*) – The file uploaded by the user

Returns A thin wrapper around the file and provider

Return type flask_store.file_wapper.FileWrapper

3.1.7 flask_store.stores.s3

AWS Simple Storage Service file Store.

Example

```
from flask import Flask, request
from flask.ext.Store import Backend, Store
from wtforms import Form
from wtforms.fields import FileField

class FooForm(Form):
    foo = FileField('foo')

app = Flask(__app__)
app.config['STORE_PROVIDER'] = 'flask_store.stores.s3.S3Store'
app.config['STORE_S3_ACCESS_KEY'] = 'foo'
app.confog['STORE_S3_SECRET_KEY'] = 'bar'

store = Store(app)

@app,route('/upload')
def upload():
    form = FooForm()
    form.validate_on_submit()

    backend = Backend()
    backend.save(form.files.get('foo'))
```

class `flask_store.stores.s3.`**S3GeventStore**(*destination=None*)

A Gevent Support for `S3Store`. Calling `save()` here will spawn a greenlet which will handle the actual upload process.

save(*file*)

Acts as a proxy to the actual save method in the parent class. The save method will be called in a `greenlet` so `gevent` must be installed.

Since the origional request will close the file object we write the file to a temporary location on disk and create a new `werkzeug.datastructures.FileStorage` instance with the stram being the temporary file.

Returns Relative path to file

Return type str

class `flask_store.stores.s3.`**S3Store**(*destination=None*)

Amazon Simple Storage Service Store (S3). Allows files to be stored in an AWS S3 bucket.

REQUIRED_CONFIGURATION = ['STORE_S3_ACCESS_KEY', 'STORE_S3_SECRET_KEY', 'STORE_S3_BUCKET',

Required application configuration variables

static app_defaults(*app*)

Sets sensible application configuration settings for this provider.

Parameters **app** (*flask.app.Flask*) – Flask application at init

bucket (*s3connection*)
Returns an S3 bucket instance

connect ()
Returns an S3 connection instance.

exists (*filename*)
Checks if the file already exists in the bucket using Boto.

Parameters **name** (*str*) – Filename to check its existence

Returns Whether the file exists on the file system

Return type bool

join (**parts*)
Joins paths into a url.

Parameters ***parts** – List of arbitrary paths to join together

Returns S3 save joined paths

Return type str

save (*file*)
Takes the uploaded file and uploads it to S3.

Note: This is a blocking call and therefore will increase the time for your application to respond to the client and may cause request timeouts.

Parameters **file** (*werkzeug.datastructures.FileStorage*) – The file uploaded by the user

Returns Relative path to file

Return type str

3.2 Change Log

3.2.1 0.0.2 - Alpha

- Feature: FileStore wrapper around provider files
- Bugfix: S3 url generation

3.2.2 0.0.1 - Alpha

- Feature: Local File Storage
- Feature: S3 File Storage
- Feature: S3 Gevented File Storage

3.3 Contributors

Without the work of these people or organisations this project would not be possible, we salute you.

- Soon London: http://thisissoon.com | @thisissoon

- Chris Reeves: @krak3n
- Greg Reed: @peeklondon

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

f