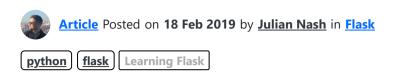
Sending files with Flask | Learning Flask Ep. 14

How to send and allow users to download files with Flask



Allowing users to download files from your website of application is an often required feature of any website or application and Flask provides us with some useful function to do so.

In this example, we're going to allow our users to download 3 types of files, images, CSV's and PDF's simply by accessing a route and providing a unique id to the resource.

Let's get started.

Flask imports

First up, we're going to need some imports from flask. Go ahead and import the following:

```
from flask import send_file, send_from_directory, safe_join, abort
```

- send_file allows us to send the contents of a file to the client
- send_from_directory allows us to send a specific file from a directory (Recommended)
- safe_join allows us to safely join a filename with a file/directory path
- abort allows us to abort a request and return an HTTP status code of our choosing

Variable rules

Before we jump in and create any routes, I want to quickly discuss variable rules which we've touched on before.

Variable rules allow values to be passed into the URL using <this_syntax> and allows us to work with variable data coming in via the URL.

Although not a necessity, Flask provides us with some useful converters to add an additional layer of validation to any values soming in via the URL.

We an use converters in our URL routes like so:

```
@app.route("/get-image/<image_name>") # No converter (defaults to string)
@app.route("/get-image/<int:image_number>") # Integer
@app.route("/get-image/<uuid:image_uuid>") # uuid
```

More from this series

Application factory
pattern
Flask MethodView
Flask Custom
Decorators
uWSGI Decorators
Flask before after
request
uWSGI Introduction
Flask and Docker
Deploy Flask to a VM
Web Image
Optimization

Flask Task Queues

Full list of variable rules:

Converter Function

Accepts any text without a slash (Default)

int Accepts positive integers

float Accepts positive floating point values

path Like string but also accepts slashes

Accepts UUID strings (Universally unique identifier) (e.g 118bc9c3-1af4-4d46-87a1-266db2e64e7a)

Using any of the converters listed above will convert the incoming variable into it's related type.

For example, if you define a url with <int:some_integer>, Flask will try to convert it into an integer, <path:path_to_some_file> will allow a path like string, including slashes etc..

Contents

Flask imports
Variable rules
Full list of variable
rules:
Directory structure

App config
Send from directory
File path in the URL
Send file and safe join
Related items

Flask Request Object Flask HTTP Methods Flask Error Handling <u>Flask Message</u> <u>Flashing</u> Flask Session Object

Recommended books

Python Basics <u>Advanced Python</u> Flask Web <u>Development</u> Django Web <u>Development</u> Python Machine <u>Learning</u> <u>Python Testing</u>

Recommended learning

Directory structure

Like many other important application configuration variables, we're going to add 3 new entries to our app.config object, each with a path to the directories we've created to hold the files we want to make available for our users.

But before we do so, we're going to create some new directories and add some files for our users to download:

- We're first going to create a new directory inside our static directory called client
- Inside of the client directory, we'll create 3 more directories, img, csv and pdf
- We'll then place 2 of each file type in their parent folders.

Note - Feel free to use dirrefent file & directory names, just be sure to update the examples in this guide with your own names

Our applications directory/file structure now looks like this (pay attention to the client directory in the static directory):

(i) × арр — арр — __init__.py - admin_views.py static --- client — csv -- sales_report.csv users.csv — img ── 001.jpg └── 002.jpg └─ pdf — 202de685-1dcb-4272-9aff-3dc10b65ef77.pdf └── 7471eaf0-f85a-48b9-8450-4ccb5d493210.pdf CSS └─ style.css - img — flask.png └─ uploads - YT-THUMB.png — post-img.png pythonise_favicon.png js └─ app.js templates --- admin dashboard.html └─ templates admin_template.html macros └─ input_macros.html public guestbook.html - jinja.html - profile.html - sign_up.html templates public_template.html — upload_image.html └─ views.py config.py requirements.txt run.py

<u>Pluralsight</u> <u>Treehouse</u>

> **Find Data** Courses

Free comparison tool for finding Machine Learning courses online.

Coursary

Open

Now that we've got our directories and files in place, let's update our app.config.

App config

We're going to create 3 new entries in our app.config object, each containing an absolute path to their corresponding directories:

```
# The absolute path of the directory containing images for users to download
app.config["CLIENT_IMAGES"] =
"/mnt/c/wsl/projects/pythonise/tutorials/flask_series/app/app/static/client/img"

# The absolute path of the directory containing CSV files for users to download
app.config["CLIENT_CSV"] =
"/mnt/c/wsl/projects/pythonise/tutorials/flask_series/app/app/static/client/csv"

# The absolute path of the directory containing PDF files for users to download
app.config["CLIENT_PDF"] =
"/mnt/c/wsl/projects/pythonise/tutorials/flask_series/app/app/static/client/pdf"
```

Now that we've updated our app config, let's go ahead and create our routes (I'd recommend using a config file for this which you can read more about here).

Send from directory

The send_from_directory function is the recommended secure way to allow a user to download a file from our application.

Let's create our first route and discuss it after:

```
@app.route("/get-image/<image_name>")
def get_image(image_name):
    try:
        return send_from_directory(app.config["CLIENT_IMAGES"],
filename=image_name, as_attachment=True)
    except FileNotFoundError:
        abort(404)
```

Let's step through what we've done:

We're using <image_name> in the URL and expect to receive the filename of the image without any slashes. As we haven't set a variable rule, Flask will default to string and not allow any slashes.

```
@app.route("/get-image/<image_name>")
```

Tip - As a reminder, if you replaced <image_name> with <path:image_name> and a user went to /get-image/path/to/the/image.png, the my_image variable would be path/to/the/image.png, so use with caution.

We then pass the image_name string to the get_image() function.

```
def get_image(image_name):
```

We setup a try & except block to catch if the filename isn't found on the server by using the FileNotFoundError handler.

```
try:
    return send_from_directory(app.config["CLIENT_IMAGES"],
filename=image_name, as_attachment=True)
except FileNotFoundError:
    abort(404)
```

Inside the try: block, we call the send_from_directory function and pass it 3 arguments:

- app.config["CLIENT_IMAGES"] The path to the directory containing the images we're allowing our users to download
- filename=image_name The image_name variable passed in from the URL
- as_attachment=True Allows the client to download the file as an attachment
- send_from_directory is then returned

Inside the except FileNotFoundError: block, we call abort() and pass it an HTTP status code, a 404 in the case that the file doesn't exist.

```
abort(404)
```

If you now go to either http://127.0.0.1:5000/get-image/001.png or http://127.0.0.1:5000/get-image/002.png, you'll instantly download the file!

If you try a filename that doesn't exist, you'll get a Not Found error in your browser.

Let's setup our remaining 2 routes to serve CSV's and PDF's.

You'll notice these 2 routes are very similar to the first, with the addition of the filename variable.

CSV route:

```
@app.route("/get-csv/<csv_id>")
def get_csv(csv_id):
    filename = f"{csv_id}.csv"

    try:
        return send_from_directory(app.config["CLIENT_CSV"], filename=filename,
as_attachment=True)
    except FileNotFoundError:
        abort(404)
```

PDF route:

```
@app.route("/get-pdf/<pdf_id>")
def get_pdf(pdf_id):
    filename = f"{pdf_id}.csv"

    try:
        return send_from_directory(app.config["CLIENT_PDF"], filename=filename,
as_attachment=True)
    except FileNotFoundError:
        abort(404)
```

Both router are identical apart from the addition of their corresponding file extensions in the filename variable, where we've just used an f string to append the extension to the filename.

We've hard coded the extension this way as we're only allowing that type of file extension from their given route. You could of course omit it and ask the user to provide the file extension too.

File path in the URL

You may want a nested directory structure within your trusted base directory, where users can provide a path to a file in the URL to retrieve a file.

Let's say reports is our trusted base directory, containing several sub-directories and files, like so:

```
app
├─ __init__.py
  - admin_views.py
  - static
    ├─ client
        └─ reports
            --- 2017
               — feb
                   L— sales
                       └─ sales_report.csv
                 — jan
                   L— sales
                       sales_report.csv
             - 2018
                — feb
                   └── sales
                       sales_report.csv
                 – jan
                   L— sales
                       sales_report.csv
              - 2019
                 — feb
                   └── sales
                       └─ sales_report.csv
                 – jan
                   L— sales
                       sales_report.csv
```

Without using a database, we can create a dynamic system of URL's and allow users to provide a path to a file.

Let's create a new route and put this into practice, allowing our user to download a report by providing a path in the URL.

First up, we'll add our reports directory to our app.config:

```
app.config["CLIENT_REPORTS"] =
"/mnt/c/wsl/projects/pythonise/tutorials/flask_series/app/app/static/client/repo
```

Now we'll create the route:

```
@app.route("/get-report/<path:path>")
def get_report(path):
    try:
        return send_from_directory(app.config["CLIENT_REPORTS"], filename=path,
as_attachment=True)
    except FileNotFoundError:
        abort(404)
```

We're doing exactly the same as above, with the exception of adding the path prefix to the URL variable.

```
@app.route("/get-report/<path:path>")
```

The path should be relative from the reports directory saved in our app.config!

If you were to go to /get-report/2018/feb/sales/sales_report.csv, the file would be downloaded. Likewise any non-existent filenames would throw a 404 error.

Send file and safe join

The send_file function is another way to allow users to download and directly access files on your server, however it's not recommended for any application that may take a filename from user sources.

```
Tip - Always use send_from_directory where possible.
```

The reason? send_file will happily return any file from a specified path! I'm sure you wouldn't want users to be able to downlaod any file from your application at their own will. If you do intend on using send_file, make sure your input source is trusted.

Let's setup a route to show send_file in action, using Flask's safe_join function:

```
@app.route("/get-csv/<path:filename>")
def get_csv(filename):
    safe_path = safe_join(app.config["CLIENT_CSV"], filename)

try:
    return send_file(safe_path, as_attachment=True)
    except FileNotFoundError:
        abort(404)
```

We use safe_join and pass it 2 arguments:

- The TRUSTED base directory
- The UNTRUSTED path to the file

This function will safely join the base directory and zero or more pathnames/filenames and return it to the safe_path variable.

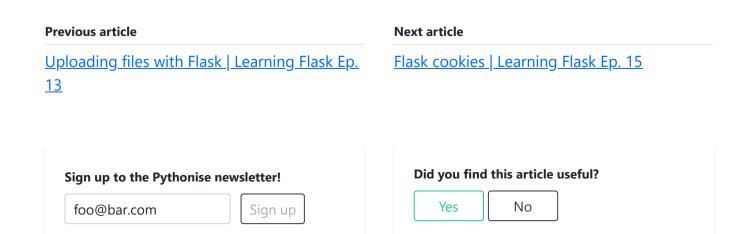
Again, you can send files this way but it's recommended to use send_from_directory

We then call send_files and pass it the safe_path along with as_attachment=True to allow the user to download the file.

Read more about sending files in Flask over at the official documentation, linked here

Last modified ⋅ 28 Feb 2019

Shared Hosting from \$1.44/mo Ad Enjoy Zero Downtime! Super Fast Sowebsite Builder Incl. 24/7 Support. Namecheap Learn More



Related items



Julian Nash · a year ago in Flask

Application factory pattern | Learning Flask Ep. 30

Building scalable Flask applications from the start using the application factory pattern, blueprints ...

<u>python</u> <u>flask</u> <u>Article</u> Learning Flask



Julian Nash · 2 years ago in Flask

Building & designing better APIs with Flask's MethodView | Learning Flask Ep. 29

Drop the @app.route decorator and write scalable, readable and better maintainable code in your ...

<u>python</u> <u>flask</u> <u>Article</u> Learning Flask



Julian Nash · 2 years ago in Python

Minifying static assets and compiling SCSS to CSS with Python

Using the sass, rcssmin and rjsmin Python packages to optimize web assets, including Bootstrap

<u>python</u> <u>flask</u> <u>Article</u>



Julian Nash · 2 years ago in Flask

Custom Flask decorators | Learning Flask Ep. 28

Using Python decorators to add another layer of functionality to Flask routes

<u>python</u> <u>flask</u> <u>Article</u> Learning Flask

This work is licensed under a <u>Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License</u>