Uploading files with Flask | Learning Flask Ep. 13

Posting, checking and validating file uploads with Flask

Article Posted on **14 Feb 2019** by **Julian Nash** in **Flask**

flask python Learning Flask

Uploading files to the server is often a requirement of a website or web application. Thankfully, Flask makes this relitively simple for us with a few useful functions.

We're using Bootstrap 4 CSS in this example but feel free to use any other CSS library, use your own or skip the styling completely.

Let's get started.

Create a new route

We'll start by creating a new route which we'll use to render a template containing a form, which users can use to upload an image.

> Tip - You'll need to import `render_template` from `flask` if you haven't already

We'll give the route the URL of `/upload-image`:

app/app/views.py

```
@app.route("/upload-image", methods=["GET", "POST"])
def upload_image():
    return render_template("public/upload_image.html")
```

We'll be making a `POST` request to the server, so we've added `methods=["GET", "POST"]` to the route.

Upload form

Now we need to create our HTML template. We'll call it `upload_image.html` and place it in the `templates/public` directory.

Go ahead and add the following:

app/app/templates/public/upload_image.html

More from this series

Application factory

Contents

Create a new route

Recommended books

Recommended learning


```
{% extends "public/templates/public_template.html" %}

{% block title %}Upload{% endblock %}

{% block main %}

<div class="container">
  <div class="row">
    <div class="col">

      <h1>Upload an image</h1>
      <hr>

      <form action="/upload-image" method="POST" enctype="multipart/form-data">

        <div class="form-group">
          <label>Select image</label>
          <div class="custom-file">
            <input type="file" class="custom-file-input" name="image"
id="image">
            <label class="custom-file-label" for="image">Select image...
</label>
          </div>
        </div>

        <button type="submit" class="btn btn-primary">Upload</button>

      </form>

    </div>
  </div>
</div>

{% endblock %}
```


e've created a new child template containing a form with a single input, a file browser.

> - When uploading images via a form with Flask, you must add the `enctype` attribute to the rm with the value `multipart/form-data`

ow that we have our form and file browser, we can move on to handling the upload in our ute.

ccessing files in a route

access a file being posted by a fowm, we use `request.files` provided by the `request` object.

need to import `request` from flask. We'll also go ahead and import `redirect` too.

```
from flask import request, redirect
```

ahead and refactor the `/upload-image` route to the following:

p/app/views.py

```
@app.route("/upload-image", methods=["GET", "POST"])
def upload_image():

    if request.method == "POST":

        if request.files:

            image = request.files["image"]

            print(image)

            return redirect(request.url)


    return render_template("public/upload_image.html")
```

- We're veryfying the request method is `POST` with `if request.method == "POST":`
- We then veryfy if the request contains files with `if request.files:`
- We then store the file as a variable called image using `image = request.files["image"]`

Using `request.files["image"]`, we're able to access the file from the form with the attribute `name="image"`

Adding another file input is as simple as creating another file input field in the HTML form and providing a different value in the `name` attribute.

For example, creating another file input with the attribute `name="image_2"` could then be accessed in Flask with `request.files["image_2"]`

Printing the `image` variable you'll see:

```
<FileStorage: 'example.png' ('image/png')>
```

You'll notice the `FileStorage` class, followed by the filename and the type of file.

FileStroage is a wrapper class around incoming files provided by Werkzeug, Flask's underlying HTTP library which handles incoming request data.

Flask stores incoming file uploads in the webservers memory (If the files are small), otherwise it will store them in a temporary location.

Saving files

We'll start with the quickest and easiest way to save a file.

You'll need the `os` library. Go ahead and import it:

app/app/views.py

```
import os
```

We should specify a directory to save our uploaded images which we'll add to our `app.config` object. You don't have to do this but it's best practice.

Either create a variable in your config file with `IMAGE_UPLOADS = /path/to/uploads/folder` or asssign it directly to `app.config["IMAGE_UPLOADS"]`.

> TIp - You should provide a complete path, making sure any directories in the path exist

In this example, we're going to assign the `IMAGE_UPLOADS` config attribute in our app but I'd recommend you create it in your app config file.

app/app/views.py

```
app.config["IMAGE_UPLOADS"] =
"/mnt/c/wsl/projects/pythonise/tutorials/flask_series/app/app/static/img/uploads
```

As you can see, we have a long but complete path!

To save the file, we simply call `image.save()` and join the path to the uploads folder with the filename using `os.join()`:

app/app/views.py

```
image.save(os.path.join(app.config["IMAGE_UPLOADS"], image))
```

Our route now looks like this:

app/app/views.py

```
from flask import request, redirect
import os

app.config["IMAGE_UPLOADS"] =
"/mnt/c/wsl/projects/pythonise/tutorials/flask_series/app/app/static/img/uploads


@app.route("/upload-image", methods=["GET", "POST"])
def upload_image():

    if request.method == "POST":

        if request.files:

            image = request.files["image"]

            image.save(os.path.join(app.config["IMAGE_UPLOADS"],
image.filename))

            print("Image saved")

            return redirect(request.url)

    return render_template("public/upload_image.html")
```

We use `image.filename` to access the filename of the image and join that with the path to the uploads folder with `os.join()`.

Save the file and upload an image to see it in action.

Securing file uploads

At this point, a user could upload any kind of file of any filesize, which is dangerous..

> Tip - NEVER TRUST USER INPUT

To mitigate any damage our application might receive from a malicius actor or user error, we should consider the following:

- Ensuring the file has a name
- Ensuring the file type is allowed
- Ensuring the filename is allowed
- Ensuring the filesize is allowed

Let's start with the filename.

Ensuring the file has a filename is a simple fix:

app/app/views.py

```
if image.filename == "":
    print("No filename")
    return redirect(request.url)
```

To ensure the type of file is allowed, we should create a set of allowed extensions in our `app.config`.

We'll just stick to image extensions for now but you'll need to modify this to allow other file types.

Go ahead and add the following:

app/app/views.py

```
app.config["ALLOWED_IMAGE_EXTENSIONS"] = ["JPEG", "JPG", "PNG", "GIF"]
```

This declares we're only going to accept 4 file extensions for image uploads.

We should create a function that we can call to confirm this:

app/app/views.py

```
def allowed_image(filename):

    # We only want files with a . in the filename
    if not "." in filename:
        return False

    # Split the extension from the filename
    ext = filename.rsplit(".", 1)[1]

    # Check if the extension is in ALLOWED_IMAGE_EXTENSIONS
    if ext.upper() in app.config["ALLOWED_IMAGE_EXTENSIONS"]:
        return True
    else:
        return False
```

Ensuring the filename itself isn't dangerous is probably even more important. Luckily for us, Werkzeug provides a handy function called `secure_filename` that we can call to return a secure filename.

First of all we need to import it:

app/app/views.py

```
from werkzeug.utils import secure_filename
```

We can now call it to return a secure filename of our file:

app/app/views.py

```
filename = secure_filename(image.filename)
```

Lastly, we need to modify `image.save()` to include the safe filename:

app/app/views.py

```
image.save(os.path.join(app.config["IMAGE_UPLOADS"], filename))
```

Putting everything together, our app now looks like this:

app/app/views.py

```
from flask import request, redirect
from werkzeug.utils import secure_filename

import os

app.config["IMAGE_UPLOADS"] =
"/mnt/c/wsl/projects/pythonise/tutorials/flask_series/app/app/static/img/uploads

app.config["ALLOWED_IMAGE_EXTENSIONS"] = ["JPEG", "JPG", "PNG", "GIF"]

def allowed_image(filename):

    if not "." in filename:
        return False

    ext = filename.rsplit(".", 1)[1]

    if ext.upper() in app.config["ALLOWED_IMAGE_EXTENSIONS"]:
        return True
    else:
        return False

@app.route("/upload-image", methods=["GET", "POST"])
def upload_image():

    if request.method == "POST":

        if request.files:

            image = request.files["image"]

            if image.filename == "":
                print("No filename")
                return redirect(request.url)

            if allowed_image(image.filename):
                filename = secure_filename(image.filename)

                image.save(os.path.join(app.config["IMAGE_UPLOADS"], filename))

                print("Image saved")

                return redirect(request.url)

            else:
                print("That file extension is not allowed")
                return redirect(request.url)

    return render_template("public/upload_image.html")
```

Lastly, we should ensure the file is of an acceptable filesize.

Just like we did with specifying the allowed image extensions in the app config. We can do the same with the maximum filesize using the default `MAX_CONTENT_LENGTH` config variable.

Let's set our maximum filesize at around 50 megabytes:

app/app/views.py

```
app.config['MAX_CONTENT_LENGTH'] = 50 * 1024 * 1024
```

This setting will apply globally to all uploads sent to your application, which may or may not be ideal.

Limiting file upload size

I've been unable to find a way to read the filesize using the Flask or Werkzeug utilities, so had to find another creative way.

There may be instances where you need users to upload different file types, all with different filesize restrictions.

An alternative to using `MAX_CONTENT_LENGTH` is to send the filesize as a cookie along with the file, validate the filesize and then decide whether to save the file of not.

In order to achieve this, we're going to do the following:

- Create a JavaScript function that saves the filesize as a cookie
- Set a maximum filesize limit for images in the app config
- Create a function to validate the image filesize

Let's create the JavaScript function to listen for an `oninput` event and attach it to the file `input` field:

```
<script>

  function filesize(elem){
    document.cookie = `filesize=${elem.files[0].size}`
  }

</script>
```

We also need to attach the `oninput` event to the `input` field and call the function:

```
<input type="file" class="custom-file-input" name="image" id="image"
oninput="filesize(this);">
```

Now, when the user changes the input value, a cookie is saved and send to our app when the form is submitted.

Let's set a `MAX_IMAGE_FILESIZE` in our app config:

app/app/views.py

```
app.config["MAX_IMAGE_FILESIZE"] = 0.5 * 1024 * 1024
```

We've set it at around 500,00 bytes for testing purposes.

Next up, we'll need to create a function to validate the filesize:

app/app/views.py

```
def allowed_image_filesize(filesize):

    if int(filesize) <= app.config["MAX_IMAGE_FILESIZE"]:
        return True
    else:
        return False
```

Cookies come in as strings, so we pass the `filesize` cookie to the `int()` function to convert it.

We access cookies using `request.cookies`, a dictionary like object which we can extract values by key.

Finally, let's grab the cookie and call the `allowed_image_filesize` function, passing the value to it:

app/app/views.py

```
if "filesize" in request.cookies:

    if not allowed_image_filesize(request.cookies["filesize"]):
        print("Filesize exceeded maximum limit")
        return redirect(request.url)
```

Our finished app now looks like this:

app/app/views.py

```
from flask import request, redirect
from werkzeug.utils import secure_filename

import os

app.config["IMAGE_UPLOADS"] =
"/mnt/c/wsl/projects/pythonise/tutorials/flask_series/app/app/static/img/uploads

app.config["ALLOWED_IMAGE_EXTENSIONS"] = ["JPEG", "JPG", "PNG", "GIF"]
app.config["MAX_IMAGE_FILESIZE"] = 0.5 * 1024 * 1024

def allowed_image(filename):

    if not "." in filename:
        return False

    ext = filename.rsplit(".", 1)[1]

    if ext.upper() in app.config["ALLOWED_IMAGE_EXTENSIONS"]:
        return True
    else:
        return False


def allowed_image_filesize(filesize):

    if int(filesize) <= app.config["MAX_IMAGE_FILESIZE"]:
        return True
    else:
        return False


@app.route("/upload-image", methods=["GET", "POST"])
def upload_image():

    if request.method == "POST":

        if request.files:

            if "filesize" in request.cookies:

                if not allowed_image_filesize(request.cookies["filesize"]):
                    print("Filesize exceeded maximum limit")
                    return redirect(request.url)

                image = request.files["image"]

                if image.filename == "":
                    print("No filename")
                    return redirect(request.url)

                if allowed_image(image.filename):
                    filename = secure_filename(image.filename)

                    image.save(os.path.join(app.config["IMAGE_UPLOADS"],
filename))

                    print("Image saved")

                    return redirect(request.url)

                else:
                    print("That file extension is not allowed")
                    return redirect(request.url)

    return render_template("public/upload_image.html")
```

Go ahead and try to upload an image over 500,000 bytes. It should print `"Filesize exceeded maximum limit"`